Special Session: On the Reliability of Conventional and Quantum Neural Network Hardware

Mehdi Sadi
Dept. of Electrical & Computer Engineering,
Auburn University, Auburn, AL, USA
mehdi.sadi@auburn.edu

Yi He, Yanjing Li
Department of Computer Science
University of Chicago, Chicago, IL, USA
{yiizy, yanjingl}@uchicago.edu

Mahabubul Alam, Satwik Kundu, Swaroop Ghosh
Dept. of Electrical & Computer Engineering,
The Pennsylvania State University,
University Park, PA, USA
{mxa890, satwik, szg212}@psu.edu

Javad Bahrami, Naghmeh Karimi
Dept. of Electrical & Computer Engineering,
University of Maryland Baltimore County (UMBC),
Baltimore, MD, USA
{jbahram1, nkarimi}@umbc.edu

Abstract

Neural Networks (NNs) are being extensively used in critical applications such as aerospace, healthcare, autonomous driving, and military, to name a few. Limited precision of the underlying hardware platforms, permanent and transient faults injected unintentionally as well as maliciously, and voltage/temperature fluctuations can potentially result in malfunctions in NNs with consequences ranging from substantial reduction in the network accuracy to jeopardizing the correct prediction of the network in worst cases. To alleviate such reliability concerns, this paper discusses the state-of-the-art reliability enhancement schemes that can be tailored for deep learning accelerators. We will discuss the errors associated with the hardware implementation of Deep-Learning (DL) algorithms along with their corresponding countermeasures. An in-field self-test methodology with a high test coverage is introduced, and an accurate high-level framework, so-called FIdelity, is proposed that enables the designers to evaluate DL accelerators in presence of such errors. Then, a state-of-the-art robustness-preserving training algorithm based on the Hessian Regularization is introduced. This algorithm alleviates the perturbations during inference time with negligible degradation in the accuracy of the network. Finally, Quantum Neural Networks (QNNs) and the methods to make them resilient against a variety of vulnerabilities such as fault injection, spatial and temporal variations in Qubits, and noise in QNNs are discussed.

I. Introduction

Thanks to their outstanding accuracy, Deep Neural Networks (DNNs) have received the lion's share of attention in recent years, and accordingly have been deployed in a wide range of safety-critical applications such as biometric security, autonomous vehicle, healthcare, financial planning, and so on [1]–[4]. Due to the computational power greediness of Deep Neural Networks, and the resource constraints of the CPU- and GPU-based platforms used for running DNN algorithms, substantial effort has been put into efficient implementations of DNN hardware accelerators in recent years to improve their performance and power consumption [5]–[8].

978-1-6654-1060-1/22/$31.00 ©2022 IEEE

Considering the application of DNN accelerators in safety-critical applications, preserving their integrity against the faults that are unintentionally or maliciously injected is of utmost importance. In practice, such faults may affect the operation of DNNs and result in a different outcome, e.g., the input can be classified incorrectly. Theoretically, DNNs are supposed to be highly error-tolerant due to the existing redundancy in their network structure [9], [10]. However, their accuracy starts to degrade when errors exceed their inherent tolerable threshold. As discussed in [11]–[14], the fault-induced accuracy decline of DNNs can cause major problems in critical applications. Accordingly, detection and possibly recovery of such errors need to be taken into account to be able to use DNNs in critical applications. For example, to preserve road safety, based on the ISO26262 [15] standard, at least 99% of faults should be detected.

Information redundancy (e.g. error correction codes) is widely used for protecting memory cells in critical applications, yet not suitable for the execution paths. Area redundancy (e.g., Triple Modular Redundancy), and timing redundancy schemes (where each operation is repeated in time) have been traditionally used to recover Single Event Upsets (SEUs). However, they impose high area and delay overhead respectively; thus not applicable to DNNs. On the other hand, the imprecise nature of NN computations introduces the possibility of approximate fault tolerance [10]. Thereby, it is essential to understand the impact of faults injected in different parts of DNN hardware to be able to tailor efficient and low overhead fault tolerance techniques for these architectures. Accordingly, this paper focuses on the efficient state-of-the-art reliability enhancement schemes for deep learning accelerators.

The rest of this paper is organized as follows. Sec. II discusses a functional in-filed self-test method to detect the DNN failures during its run-time. Then, it introduces a framework to analyze the resiliency of DNN hardware. In Sec. III the

effects of hardware hazards on the inference accuracy of DNN accelerators are discussed. Sec. IV presents a robustness-aware training algorithm that takes into account hardware associated uncertainties throughout the learning phase. Then in Sec. V Quantum Neural Networks (QNNs) are discussed and the schemes to address robustness concerns in these structures are presented. Finally, Sec. VI concludes the paper.

II. Resilience Techniques for Deep Learning Accelerators

Hardware error resilience is a top priority for deep learning (DL) accelerators, as hardware errors can generate various unexpected outcomes such as system crashes, silent data corruptions (SDC), INF/NaN values, and more. These unexpected outcomes can compromise the output quality of DL workloads, and can even lead to life-critical threats (e.g., in self-driving car applications). Although there exist a few studies on hardware errors in DL accelerators [16], [17], [17]–[20], they are largely limited to memory errors only. This is insufficient, because logic components (i.e., sequential elements and combinational logic) can also have a significant impact on the overall reliability. For example, based on our analysis, for a DL accelerator without any resilience support, the FIT (failure in time) rate of flip-flops (FFs) (e.g., >9.5 in Nvidia's open-sourced DL accelerator called NVDLA) is significantly higher than the automotive safety requirement (<0.2), even just as a result of random transient errors that occur infrequently [21]. Permanent hardware failures in the logic portion of an accelerator will further exacerbate the overall FIT rate.

It is essential to understand and mitigate both permanent and transient errors in the logic portions of DL accelerators. This is the focus of our work, which uniquely combines the knowledge of testing, hardware error resilience, computer architecture, and machine learning to enable resilient DL accelerators.

A. Functional In-Field Self-Test for DL Accelerators to Mitigate Permanent Hardware Failures

For permanent hardware failures (such as early-life failures, circuit aging, and manufacturing defects) which are a major reliability concern [22], we present an efficient in-field self-test approach, which allows a DL accelerators to periodically test itself in the field to detect (or even predict) these failures [23]. Our technique generates high-quality functional in-field self-tests specifically targeting DL accelerators, which is crucial to ensure that the safety and/or reliability requirements are met for any given application.

Compared to structural tests, the main advantage of functional tests is that they do not require structural test support (e.g., CASP [24], [25] or Logic BIST [26]) in the hardware. On the other hand, a well-known challenge of functional tests is that the test coverage is typically much lower than that of structural tests, due to both observability and controllability constraints. Fortunately, for in-field self-tests that are applied for the purpose of detecting permanent hardware failures, it is sufficient to target only the faults that can affect application correctness to achieve high functional test coverage. However, this is still a significant challenge because high quality functional tests must be applied to complex DL accelerators under and various system-level constraints (e.g. test time and power, etc.).

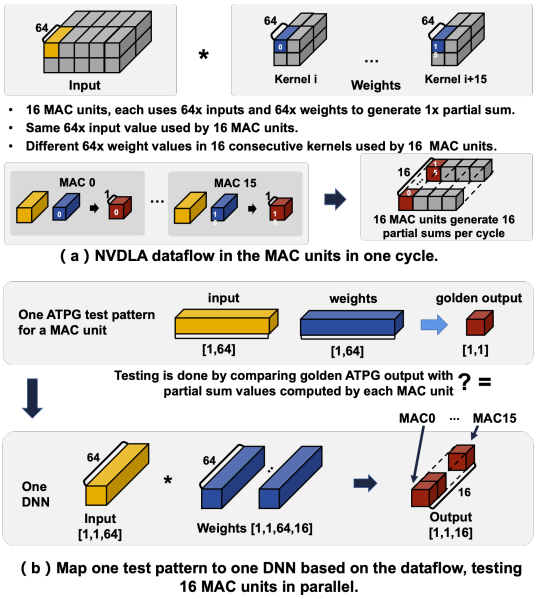


Figure 1. Our functional in-field self-test generation technique for DL accelerators, demonstrated using the Multiply-Accumulate (MAC) units in NVDLA as an example.

Our technique achieves high functional test coverage by taking advantage of special architectural characteristics and application properties of DL accelerators. For the compute units (i.e., hardware modules that perform computations to transform data, such as the multiply-accumulate unit), we first use combinational ATPG (automatic test pattern generation) to generate test patterns with high test coverage, and then map the ATPG patterns to one or more equivalent deep neural networks (DNNs) that can be directly executed on the accelerator (an example of how the mapping is done is shown in Fig 1). This is possible because: (1) these units generally do not contain complex sequential logic, so combinational ATPG yields high test coverage; (2) given that the dataflow/reuse algorithms of a DL accelerator are well defined, the inputs of individual compute units can be mapped to the primary inputs of the accelerator through reverse-engineering of the dataflow.

For the control units (i.e., hardware modules that are solely responsible for data movement, such as the input/weight sequencing units), we leverage the property that typically only one or a few fixed DNNs are deployed at a time in many application domains. Thus, it is sufficient to target only the faults that can directly affect the correctness of the DNNs that are currently deployed. We are able to mathematically prove that, by executing different layers of a given DNN using input and weight tensors with linearly-independent columns, 100% test coverage is achieved for all single-variable-type control fault models (i.e., single or multiple faults that only affect the control logic associated with a single variable type, such as inputs or weight), out of all control faults that can affect the

correctness of the given DNN.

We apply our technique using NVDLA as a case study to demonstrate its efficacy. Our results show that:

(1) For compute units, 99.9% single stuck-at functional test coverage is achieved. For the control units, 100% functional test coverage can be achieved for all single-variable-type fault models.

(2) The in-field functional self-test time ranges from 1.13-16.84 ms for various representative DNNs (including GoogleNet, Yolo, DenseNet and EfficientNet), and the test storage is <600MB.

With such high test coverage and low costs, our approach is effective and practical in various use scenarios to achieve high levels of reliability/safety requirements. These functional tests can be applied during boot-up, reset, and even concurrently with normal operation by executing DNN test programs directly on a DL accelerator, without requiring any test support in the hardware.

B. FIdelity: Efficient Resilience Analysis framework for DL Accelerators

To understand and mitigate the effects of transient errors (such as soft errors and dynamic variations) in the logic portions of a DL accelerator, we present a resilience analysis framework called FIdelity [21]. FIdelity enables efficient resilience analysis for all single-cycle, single FF bit-flip errors (or multiple single-cycle bit-flips in a single register), which are the most prominent abstraction for transient errors including soft errors [27] and voltage variations [28].

A well-known resilience analysis approach is to perform large-scale fault injection experiments. However, existing fault injection techniques suffer from the following limitations: (1) RTL-Level fault injection techniques can achieve accurate results, but they require access to RTL to perform time-consuming RTL simulations, which is costly and may not even be feasible; (2) software-level techniques can produce results quickly, but they are highly inaccurate.

Our FIdelity framework overcomes the limitations of existing fault injection techniques. An overview of our FIdelity framework is shown in Fig. 2 (a), which models hardware transient errors in software with high fidelity, while requiring only a few important pieces of architecture/microarchitecture information that is pertinent to resilience analysis (e.g., reuse/scheduling algorithms, pipeline configurations, etc.). This accurate mapping is possible because we identify and leverage the following architectural properties that are commonly shared by a wide range of DL accelerators:

(1) A hardware error can only directly affect the results of the current DNN layer, then propagate to other layers. Thus, to capture the effects of an error on the final output, it is equivalent to first obtaining the effects of the error in the current DNN layer, and then determining how the faulty layer affects the final output through software simulation.

(2) The next question is: how to determine the effects of an error in the current DNN layer? It turns out that, due to the regular structure and precisely-defined dataflow architecture,

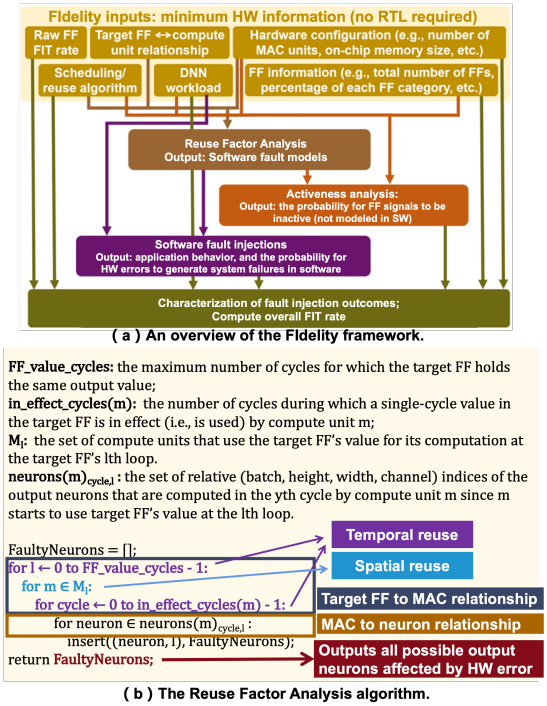


Figure 2. The FIdelity framework and the Reuse Factor Analysis algorithm.

the value stored in a datapath FF only affects a deterministic set of output neurons in the current DNN layer in each cycle. Leveraging this property, we develop an algorithm called *Reuse Factor Analysis*, demonstrated in Fig. 2 (b). Given a target FF and high-level hardware information shown in Fig. 2 (b), the Reuse Factor Analysis algorithm takes into account both spatial and temporal reuse of the FF value to return the following information: (1) the maximum number of faulty neurons that can be generated if this FF experiences a single cycle bit-flip; (2) the relative location(s) of all possible faulty neuron(s); and (3) the order in which these faulty neurons are calculated.

(3) After obtaining information about the positions of faulty neurons, the faulty values of these neurons need to be determined as well. Here, we leverage the third property of DL accelerators: datapath FFs closely match software variables. Therefore, one bit-flip in any datapath FF can always be mapped to an equivalent set of bit-flips in software so we can model the hardware error effects precisely in software fault injection.

(4) Datapath hardware errors occurring in the same pipeline stage and belonging to the same variable type (weight, input feature, output activation, and bias) exhibit the same error effects, which makes our resilience analysis approach a tractable problem.

(5) The effects of an error occurring in the control logic can also be determined, depending on the type of the control error. On the one hand, a global control error (e.g., one that causes the accelerator to incorrectly perform int16 operations even though the data precision should be FP16) almost always results in incorrect final results, crashes, or hangs. On the other hand, a local control error affects only the datapaths directly

connected to the corresponding control signal, so its effects are the union of all error effects from the connected datapaths.

We thoroughly validate our framework by applying it to NVDLA. We first obtain the complete set of accurate software fault models to capture the effects of hardware transient errors in this design. Next, we perform 60K RTL fault injection experiments using various representative DNN workloads. By manually analyzing all RTL fault injection cases that lead to non-masked outcomes, we confirm that, for the datapath, the software fault models derived using FIdelity capture the *exact* fault behaviors obtained from RTL simulations. For the control portion, FIdelity's software fault models closely match RTL results.

Using the validated FIdelity framework, we perform a large-scale resilience study on NVDLA, which consists of 46M fault injection experiments running various representative DNNs. The key results reveal many important insights. For example, we are able to quantitatively demonstrate the crucial need for resilience analysis and protection solutions for DL accelerators. Although DL workloads exhibit certain tolerance to errors, such tolerance alone cannot guarantee that a DL accelerator will meet the resilience requirement of a target application. Moreover, based on the fault injection experiments, we perform detailed analysis to understand how error magnitude, hardware design choices, data precision, and correctness metrics affect the overall resilience of the design. This knowledge can be leveraged to guide DL accelerator design decisions as well as the development of new resilience techniques.

III. Impact of Hardware and Circuit Hazards on the Reliability of Deep Learning/AI Accelerators

The major types of circuit and transistor-level runtime hazards that can impact the performance of Deep Learning accelerator hardware are, (i) dynamic power supply voltage noise and droop, (ii) circuit aging with time, and (iv) radiation-induced soft errors as shown in Fig. 3(a). Voltage noise is caused by simultaneous switching events inside the chip [29]. Since the accelerator chips will perform billions of MAC operations to correctly classify camera images or input patterns, the extensive switching inside the MAC circuits and memory units from this can cause voltage noise. The corresponding transient voltage droop can cause timing violations or bit-flips inside the accelerator. Since safety is of utmost importance, accelerators used in Autonomous Vehicles (AV) must consider such events and incorporate built-in robustness into the models against such hardware hazards. Circuit aging is a critical reliability problem in modern VLSI chips [30], [31]. Since aging is use-case dependent and cannot be accurately estimated at time 0, it is a major concern for safety-critical automotive applications. Aging can impact the weight storage SRAM modules by altering their read/write stability with time, and thus cause bit-flip errors. Although for AVs, operating at sea-level altitude, it may seem they are immune to soft errors caused by high-energy neutrons from cosmic radiation. However, as shown in detail in [32], because of weight reuse in Deep Learning, soft-errors can indeed impact the accuracy of accelerators. Hence, it is essential to embed resiliency and robustness at the training algorithm level against random bit-flips [33]–[35]. For safety-critical applications, chips with permanent stuck-at faults are generally discarded according to defective parts per million/billion (DPPM/DPPB) [36] guidelines of FuSa standards [15]. However, aging-induced in-field stuck-at faults can be a concern for applications of deep learning for navigation in safety-critical AV.

Unlike other areas, a one-size-fits-all training method for (Deep Neural Network) DNNs may not be suitable for Deep Learning used in safety-critical applications such as AVs. The DNN training algorithm needs to be revisited to incorporate resiliency and robustness in the trained model against weight perturbations that might occur in the memory and MAC modules at runtime.

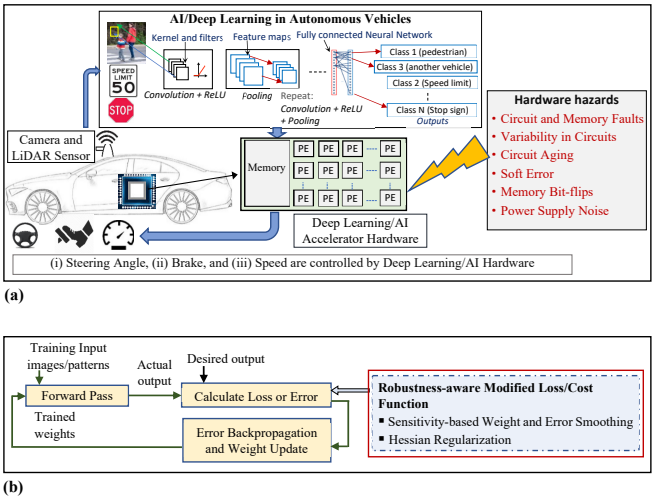


Figure 3. (a) Various circuit-level hazards can impact DNN accelerators. Unlike other domains, AI chips used in safety-critical Autonomous Vehicles must have additional resiliency. (b) Robustness-aware regularization technique can be incorporated into the conventional DNN training algorithm to enhance inference-time resiliency against hardware hazards.

IV. Robustness-aware Training of Deep Neural Networks

In conventional backpropagation, the Stochastic Gradient Descent (SGD) or ADAM algorithm converges to the local minima of the error surface with respect to the weight space [37]. As a result of this optimization objective, the error surface might be relatively steep as SGD method prefers sharp descent to the optimum solution point [38]. Hence, any perturbations in the trained weights from the optimum point might cause the prediction error to increase substantially (Fig. 4). A technique to increase the robustness of the trained Deep Learning model against weight perturbations would be to smooth the error surface with respect to the weight fluctuations across the optimum solution point in the weight space. In other words, if the plane on the hyper-surface around the trained weight space is relatively flat, then a small change in the weights or equivalent activations from MAC output will not cause large degradation in the prediction error. To increase the robustness

of Deep Learning algorithms used in the safety-critical domain against hardware hazards, the error vs. weight space must be flattened across the optimum solution point. This will ensure that the inference accuracy is not compromised if the hardware-level hazards are within a certain extent. However, conventional backpropagation and SGD cannot directly achieve this goal [37], [38].

Because of the mathematical form of the loss or cost function used in the classical backpropagation algorithm, the SGD converges to the weight solution space that minimizes loss in prediction accuracy oblivious of any hardware perturbations or hazards that might occur during inference. A resilient approach would be to select the weight configuration that not only minimizes loss but also accounts for any hardware hazards (within a certain range) as the solution or trained weights. This targeted smoothing of the solution locus and democratization of the collective importance of the weights are essential for safety-critical AI applications. This resilience-enhancing training method can be interpreted as a statistical sensitivity reduction technique where the learning of the model is distributed among the different weights and neurons, rather than a particular set of weights and neurons. Because of this training approach, the tolerance of the trained Deep Learning model against perturbations in weights and MAC outputs will be increased.

A. Regularized Training to Smoothen the Curvature of Loss Surface

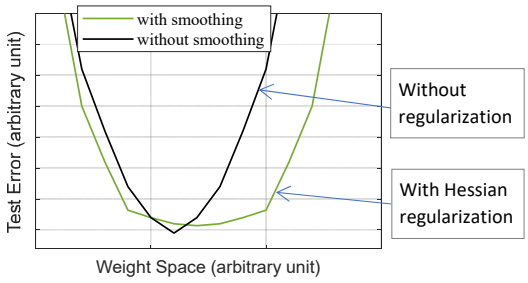


Figure 4. Visualization of the loss surface in 1-D [39]. Hessian regularization can reduce the curvature of loss surface, thus enhance the resilience of the trained model against perturbations.

The concept of regularization is well known in the machine learning community [37]. To avoid overfitting in training, the L_1 and L_2 regularizers are widely used to limit the magnitudes of the trained weight parameters. Using a similar concept, in this research the conventional backpropagation method of training Deep Learning models has been augmented with robustness-enhancing regularizers. This penalty-based regularization technique enforces smoothness or flatness of the solution (i.e., trained weights) surface. In essence, the regularizers act as a penalty term against hardware hazards (e.g., stuck-at faults in LSBs, bit-flips from circuit aging, soft-error, voltage noise, etc.), and when augmented with the regular cost function in the backpropagation equation, the training procedure jointly minimizes the original loss term as well as the extra loss/penalty caused by hardware imperfections. This method of regularization can significantly enhance the resiliency of Deep Learning models used in the safety-critical AV domain.

Because of the use of robustness-enhancing regularization techniques in the augmented backpropagation equation, the training loss can be slightly degraded compared to the conventional backpropagation method. However, this slight imperfection of the modified backpropagation approach is acceptable because in AVs Deep Learning models are deployed for identification of humans, road signs, speed limits, other vehicles, etc., hence detection details similar to biometrics (e.g., face recognition) are not required. For example, it is enough to perceive the presence of a human ahead in the road rather than identifying the exact identity of the person from a detailed analysis of the facial features. Moreover, the slight increase in training loss caused by this approach can be recovered by increasing the training epochs. As the robustness-aware augmented backpropagation algorithm can place the optimum solution at a flatter region of the loss surface, the trained Deep Learning model exhibits better resiliency to hardware hazards during deployment in the safety-critical AVs.

The theoretical basis of the proposed robustness-aware training methodology for safety-critical applications can be explained by the Taylor series expansion of the original loss function (e.g., Mean Squared Error, Maximum likelihood, etc.) used in the backpropagation algorithm. The loss function under perturbation $L(\overline{\mathbf{w}}+\overline{\Delta\mathbf{w}})$ can be expanded in Taylor series across $\overline{\mathbf{w}}$ as shown in Equation 1, where $\overline{\mathbf{w}}$ is the weight vector, $\overline{\Delta\mathbf{w}}$ is the weight perturbation, N_w is number of weights.

$$L(\overline{\mathbf{w}},\overline{\Delta\mathbf{w}}) = L(\overline{\mathbf{w}}) + \sum_{i=1}^{i=N_w} \Delta w_i \frac{\partial L(\overline{\mathbf{w}})}{\partial w_i} + \frac{1}{2} \sum_{i,j=1}^{i,j=N_w} \Delta w_i \Delta w_j \frac{\partial^2 L(\overline{\mathbf{w}})}{\partial w_i \partial w_j} \tag{1}$$

Equation 1 can be interpreted as, $L_{with\_perturbation} = L_{original} + S(\overline{\Delta\mathbf{w}})$. The term $S(\overline{\Delta\mathbf{w}})$ represents the accuracy error caused by weight fluctuations (i.e., memory bit flips caused by, soft-error, circuit aging, voltage noise, stuck-at,etc.). The perturbations caused by MAC hardware can also be modeled into equivalent weight perturbations and incorporated in this term. Since in this robustness-aware loss function the perturbation term is included, during training this term is jointly minimized with the regular loss function $L_{original}$. $S(\overline{\Delta\mathbf{w}})$ contains the Jacobian of the weights and the second derivative or Hessian matrix of the loss function with respect to the weight values. During training, this Hessian term can be calculated with classical methods such as those proposed in [40]. However, Hessian (i.e., the second-order derivative) computation per mini-batch iteration during training is computationally expensive and requires large GPU memory. In Equation 2, the optimum solution weight vector is $\overline{\mathbf{w}^*}$ that minimizes both the regular prediction loss as well as errors caused by hardware hazard induced perturbations. The modified backpropagation-based weight update equations are shown in Equation 3, where α is the learning rate, β momentum factor, ∇_w is the error gradient, and n is iteration step.

$$\overline{\mathbf{w}^*} = \arg\min_{\overline{\mathbf{w}}} L(\overline{\mathbf{w}}, \overline{\Delta \mathbf{w}}) \tag{2}$$

$$\begin{aligned}(\overline{\mathbf{w}^*})^{(n+1)} = (\overline{\mathbf{w}^*})^{(n)} - \alpha \nabla_w L((\overline{\mathbf{w}^*})^{(n)}) \\ + \beta((\overline{\mathbf{w}^*})^{(n)} - (\overline{\mathbf{w}^*})^{(n-1)}) - \gamma \nabla_w S((\overline{\mathbf{w}^*})^{(n)})\end{aligned} \tag{3}$$

B. Hessian Regularized Training at Reduced Computational Complexity

Evaluating the Hessian term of Equation 1 at backpropagation is computationally very expensive, moreover, it also requires large GPU memory. To circumvent these challenges, in this research we adopt the vector-based Hessian estimation approach proposed in [41]. Additionally, to save computation overhead, we activate Hessian regularization only after the model has been trained for several epochs as shown in Algorithm 1.

Algorithm 1 Training with Hessian Regularization in PyTorch

```
1: procedure PRE-TRAINING AND HESSIAN REGULARIZED TRAINING
2: Input: Training Data
3: Input: The Deep Learning architecture, Model
4: Input: Training Hyperparameters
5: Input: Epoch after which Hessian regularization starts, N_Reg
6: Output: Trained and regularized Model
7:
8:     for epoch = 1 to Total_Epoch do
9:         for minibatch_id = 1 to Total_minibatch do
10:            minibatch_output = Model(minibatch_input)
11:            Reset stored gradients: optimizer.zero_grad()
12:            loss = criterion(minibatch_output, target)
13:            Backpropagate: loss.backward(retain_graph, create_graph)
14:            if (epoch > N_Reg) AND ((minibatch_id mod 2) == 0) then
15:                Hessian_Trace = calculate_Hessian_trace(Model)
16:                α = 0.1*|loss| / |Hessian_Trace|
17:                loss_Hessian = α*Hessian_Trace
18:                Backpropagate: loss_Hessian.backward()
19:            end if
20:            Update weights: optimizer.step()
21:        end for
22:    end for
23: end procedure
24:
25: procedure CALCULATE_HESSIAN_TRACE
26: Input: The Deep Learning model, Model
27: Input: Total iteration, N_iter
28: Output: Hessian_Trace
29:
30:    L_gw = list of (gradient, weight) tuples for all layers of Model
31:    for k = 1 to N_iter do
32:        for n =1 to length of L_gw do
33:            v(n) = random vector of size L_gw(n) with Rademacher distribution
34:            Hv(n) = create Hessian vector product with torch.autograd.grad()
35:            Trace(k) += v(n)*Hv(n)
36:        end for
37:    end for
38:    Hessian_Trace = (Σ_{k=1}^{N_iter} Trace(k)) / N_iter
39: end procedure
```

The algorithm and pseudo code of the Hessian regularized training flow for robust AI are shown in Algorithm 1. The inputs to the Algorithm are the training data set, the model architecture and hyperparameters, and the epoch after which the Hessian Regularization will be initiated. The output will be a trained model with its loss surface desensitized against weight fluctuations. The training steps are shown in Lines 8 to 23. The total training epochs are divided into two parts, the model is trained with conventional backpropagation in the first part for the majority of the epochs. In the second part, for the last few epochs (controlled by the parameter N_{Reg} in Line 14), the computationally complex Hessian regularization is activated. In Line 13, the loss function is backpropagated and the gradients of all the weights are calculated. To ensure the calculation of the second derivative (i.e., Hessian) of the loss function, the computational graph is retained in Line 13. After training the model with normal backpropagation for up to epoch N_{reg}, the Hessian regularization is activated in Line 14. To reduce computational complexity the Hessian regularization is performed on every other minibatch iteration per epoch in Line 14. To calculate the Hessian trace (i.e., the sum of diagonal elements of the Hessian matrix) of the loss function with respect to the weights, the Hutchinson trace estimator method [41] is utilized in Line 15. The calculated sum of the Hessian trace of all layers is large and for joint minimization with regular loss term it is scaled by the factor α in Line 17. α is chosen such that the Hessian loss term in Equation 1 is within 10% of the original loss term. From our experiments, we observed that this 10% scaling yielded the best results. The formula to estimate α is shown in Line 16. This scaling is required to ensure that the original loss term is not degraded while backpropagation attempts to reduce the Hessian loss term. The gradients of Hessian loss, *loss_Hessian* are calculated in Line 18. Finally, in Line 20, weights are updated considering both the gradients of the main loss function and the Hessian loss function (when activated as in Line 14).

The Hutchinson trace estimator method [41] is described in Lines 26 to 38. In Line 30 a list, L_{gw}, containing the (gradients, weights) pairs of all layers are created. The gradients are already generated when the main backpropagation occurred in Line 13, hence no extra computational cost is necessary for this step. For each element of the tuple list, L_{gw}, the Rademacher random vector $v(n)$ is generated in Line 33. Using PyTorch's autograd function, the Hessian vector product is calculated in Line 34, followed by multiplication with $v(n)$ in Line 35. Finally, the trace is averaged over all iterations, to calculate the final Hessian trace in Line 38.

The key challenges of computational complexity arise from, (i) additional memory requirement to retain the computation graph for second backpropagation (Line 13). (ii) increased floating-point operations to calculate the gradients of the Hessian loss (i.e., *loss_Hessian* in Lines 17-18). However, since the Hessian regularization is activated in the last few epochs of training, these additional costs are acceptable, especially for safety-critical applications of AI/Deep Learning.

C. Experimental Results of Hessian Regularization

In our experiment, we used the CIFAR-10 data set for training and inference. The results of Hessian regularization (following Algorithm 1) are shown in Fig. 5. The weights were perturbed by 10% of their original value. Fault rate in Fig. 5 indicates what percentage of all the weights were perturbed by this amount. Results of Fig. 5 indicate that Hessian regularization enhances resilience and fault-tolerance of DNNs.

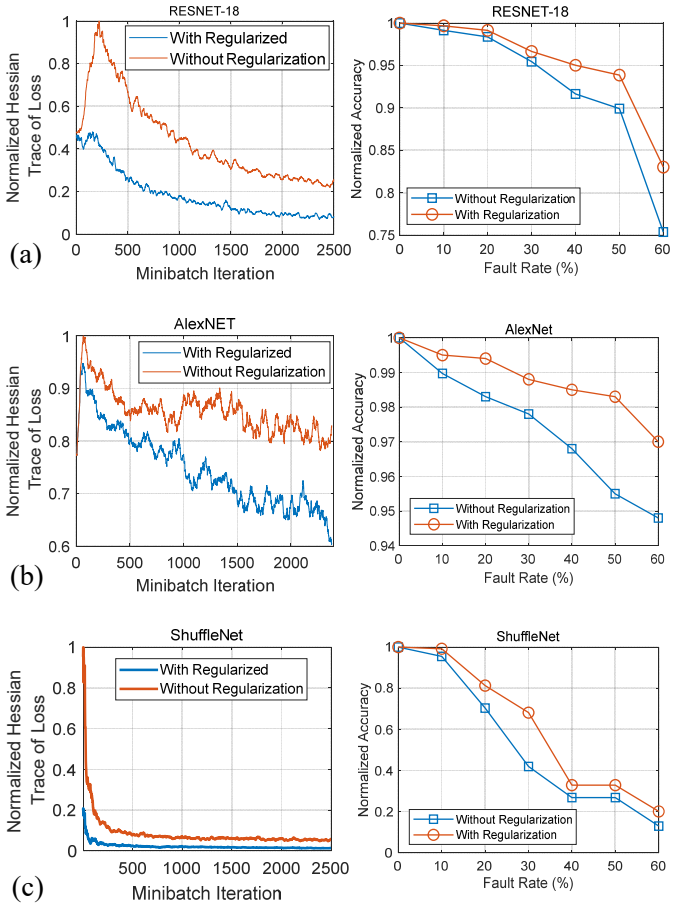


Figure 5. Hessian regularization during training enhances the robustness of DNN models at inference. Initially the DNN models are trained in conventional methods, and during the last several epochs the Hessian regularization is turned on. Hessian trace of the weight matrix decreases for regularized training, thereby increasing the robustness and resiliency of the model against hardware faults. Results for CIFAR-10 dataset on, (a) ResNet-18. (b) AlexNet. (c) ShuffleNet.

V. Resilience of Quantum Neural Network

Quantum machine learning (QML) is an emerging field that aims to develop quantum algorithms to perform conventional generative/discriminative machine learning tasks (e.g., classification, regression, etc.) [42]–[45]. One of the most promising QML models available is the Quantum Neural Network (QNN) [43], [46]–[49].

The major building block of a QNN is a Parameterized Quantum Circuit (PQC) which is a quantum circuit with tunable parameterized gates as shown in Fig. 6(a). It is often composed of successive layers of single-qubit rotations (for the purpose of exploring the search space) and multi-qubit operations (to create entanglement). The parameters of PQC can be tuned to attain desired outputs for given inputs (e.g., classifying data samples). QNN models are claimed to be more expressive compared to the classical neural networks [49], [50]. In other words, QNN models have higher capability to approximate a desired functionality compared to the classical models of similar scale (e.g., with same number of tunable parameters/weights). They also learn faster, which means that QNN models may be trained with less epochs. These encouraging theoretical studies have piqued the curiosity of a large number of application researchers. QNNs have been used in recent studies for image classification, protein classification, and drug-like molecule predictions, to name a few.

When a quantum program is executed on a quantum computer, it encounters noisy inputs and a slew of quantum-physical noise effects that combine and compound, resulting in the output of the quantum program being erroneous, varied, unstable, and, in some situations, stochastic. To assess the true potential of QNNs, we must first ask (i) how resilient they are to adversarial noise and quantum noise, and (ii) how reliable their results are when we run them on actual hardware. The work in [51] revealed that the training landscape in parameterized quantum circuits might have vanishing gradients. These locations of vanishing gradients are referred to as *barren plateaus* in the literature. Once stuck in a *barren plateau*, gradient-based optimization methods (e.g., stochastic gradient descent) may not be able to move further to train the network. In [52], the authors showed that quantum-noise could also induce *barren plateaus* in the PQC training landscape, making deep QNN training on actual hardware challenging. When tested on actual hardware, a trained QNN that performs well in a noiseless simulator performs poorly. The higher the level of noise on the hardware, the worse the performance. Because of manufacturing variations, not all qubits are created equal, and some have less noise than others [53]. As a result, when a trained QNN is run on different qubit segments of the same hardware, its performance may vary. On top of that, the quality of a qubit can vary over time. Therefore, performance of a trained QNN may vary over time on the same piece of hardware [54]. In [55], the authors demonstrated that a small amount of perturbation in the input data is enough to induce misclassification in a trained quantum classifier. An adversary can exploit these vulnerabilities to attack QML applications. Several attacks have been demonstrated already in recent academic studies [56], [57].

In this Section, we provide current developments and perspectives on robustness of QNNs against (a) adversarial input variations, (b) spatial and, (ii) temporal variations in qubits. We also discuss recent advances in noise mitigation techniques for QNN.

A. Quantum Computing Basics

Qubits, Quantum Gates, Measurements & Quantum Circuit: Unlike a classical bit, a qubit can be in a superposition state i.e., a combination of $|0\rangle$ and $|1\rangle$ at the same time. A variety of technologies exist to realize qubits such as, superconducting qubits, trapped-ions, to name a few. Quantum gates (e.g., single qubit Pauli-X gate or 2-qubit CNOT gate) modulate the state of qubits and thus perform computations. These gates can perform a fixed computation (e.g., an X gate flips a qubit state) or a computation based on a supplied parameter (e.g. the RY(θ) gate rotates the qubit along the Y-axis by θ). A two-qubit gate changes the state of one qubit (*target qubit*) based on the current state of the other qubit

(*control qubit*). For example, the CNOT gate flips the target qubit if the control qubit is in $|1\rangle$ state. A quantum circuit contains many gate operations. Qubits are measured to retrieve the final state of a quantum program.

Quantum Noise: Errors in quantum computing can be broadly classified into, (i) Coherence errors: a qubit can retain its state for a short period (coherence time). The computation needs to be done well within this limit. (ii) Gate errors: quantum gates are realized using microwave/laser pulses. It is impossible to generate and apply these pulses precisely in actual hardware making gate operations erroneous. (iii) Measurement errors: a $|0\rangle$ state qubit can be measured as $|1\rangle$ (or vice versa) due to imprecise measurement apparatus. Execution of multiple gates in parallel can lead to crosstalk errors. Since a large/deep quantum circuit accumulates more errors, a smaller circuit is always preferred for noise-resilience and reliability.

Quantum Circuit Compilation: A practical quantum computer generally supports a limited number of single and multi-qubit gates known as *basis gates* or native gates of the hardware. For instance, the current generation of IBM quantum computers have the following basis gates: ID, RZ, SX, X (single-qubit), CNOT (two-qubit). However, the quantum circuit may contain gates that are not native to the target hardware. Hence, the gates in a quantum circuit need to be *decomposed* into the basis gates before execution. Besides, the native two-qubit operation may or may not be permitted between all the two-qubit pairs. These limitations in two-qubit operations are also known as *coupling constraints*. Conventional compilers add necessary SWAP gates to meet the coupling constraints. Thus, a compiled circuit depth and gate counts can be significantly higher than the original.

Quantum Neural Network: QNN involves parameter optimization of a PQC to obtain a desired input-output relationship. QNN generally consists of three segments: (i) a classical to quantum data encoding or embedding circuit, (ii) a parameterized circuit (PQC), and (iii) measurement operations. A variety of encoding methods are available in the literature [47]. For continuous variables, the most widely used encoding scheme is angle encoding [47]–[49] where a continuous variable classical feature is encoded as a rotation of a qubit along a desired axis (X/Y/Z). For 'n' classical features, we require 'n' qubits. For example, RZ(f1) on a qubit in superposition (the Hadamard - H gate is used to put the qubit in superposition) is used to encode a classical feature 'f1' in Fig. 6(c). We can also encode multiple continuous variables in a single qubit using sequential rotations (Fig. 6(d)). States produced by a qubit rotation along any axis will repeat in 2π intervals (Fig. 6(b)). Therefore, features are generally scaled within 0 to 2π in a data pre-processing step. One can restrict the values between $-\pi$ to π to accommodate features with both negative and positive values.

The PQC consists of multiple layers of entangling operations and parameterized single-qubit rotations. The entanglement operations are a set of multi-qubit operations between the qubits to generate correlated states [48]. The following parametric single-qubit operations search through the solution space. This combination of entangling and rotation operations is referred to as a parametric layer (PL). The optimal number of PL for any given ML task is generally unknown. The problem is similar to choosing the number of hidden layers/neurons in a classical DNN. In practice, one needs to go through multiple training iterations with different number of PL's to come up with a compact network. A compact network can offer better noise-resilience/reliability, and lower latency/faster execution during inference. There is a wide variety of choices available for PL. The work in [58] analyzed 19 widely used PL architectures from literature. A widely used PL is shown in Fig. 6(a). Here, CNOT gates between neighboring qubits create the entanglement, and rotations along X & Z-axis using RX(θ) & RZ(θ) operations define the search space.

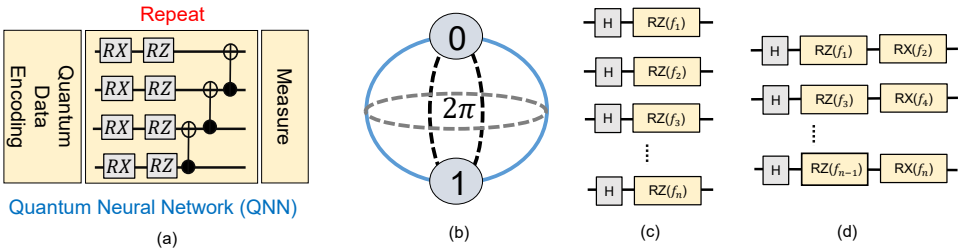
B. Robustness of QNN against Adversarial Input Variations

High-dimensional quantum systems are required to demonstrate a practical quantum advantage. Dimensionality expands exponentially with the number of qubits. Therefore, large number of entangled qubits contribute to the potential power of quantum devices over classical resources. Liu et al. [55] show that quantum classifiers are vulnerable to adversarial input perturbations. The amount of perturbation required to cause a misclassification scales inversely with dimension. This results in a trade-off between the classification algorithm's security against adversarial attacks and the quantum advantages we expect for high-dimensional problems. In [57], the authors demonstrate methods to generate adversarial samples for a quantum classifier (noisy inputs that are miss-classified by the classifier) in both white-box and black-box attack scenarios. The additional noise acts as a unitary that modifies the input state to the classifier.

C. Robustness of QNN against Adversarial Fault Injection

Multiple users can run their quantum programs at the same time (called multi-programming) if a quantum device has enough qubits. In such cases, each program will introduce some crosstalk errors in the other programs. An adversary can deliberately introduce crosstalk errors in a victim program in a multi-programming environment [59]. The accuracy of the classifier degrades significantly under attack. A simulation result in the ideal scenario (no errors) shows that the ratio of the correct outcome and the total number of shots (8192) per sample (r_μ) is close to 0.88 (on average) for the two-qubit Iris classifier. However, in experiment on ibmq_16_melbourne's Q0 and Q1 with all other qubits idle, a ratio close to 0.76 is noted given the noisy hardware.

Later, the same experiment is executed on the hardware while attacking two other qubit pairs with repeated CNOT operations (Figure 7(a)). When only pair-1 or pair-2 is attacked, the ratios are found to be 0.715 and 0.709, respectively (Figure 7(b)). The ratio further degrades to 0.69 when both pairs are attacked. The classifier misclassifies (i.e., the ratio < 0.5) 1 time in the attack-free case. It misclassifies 4 times when either pair-1 or pair-2 is attacked and 11 times when both pairs are attacked (Figure 7(b)).



40:

Figure 6. A toy Quantum Neural Network (QNN) is shown in (a). An encoder encodes classical data as quantum state. A Parametric Quantum Circuit (PQC) transforms the state. The output state is retrieved through measurements. Bloch sphere representation of a qubit is shown in (b). A qubit can be rotated along the X, Y, or Z axis. The states repeat in 2π intervals. In (c) and (d), we show angle encoding 1:1 (i.e., one continuous variable encoded in a single qubit state), and angle encoding 2:1 (i.e., two continuous variables encoded in a single qubit state), respectively.

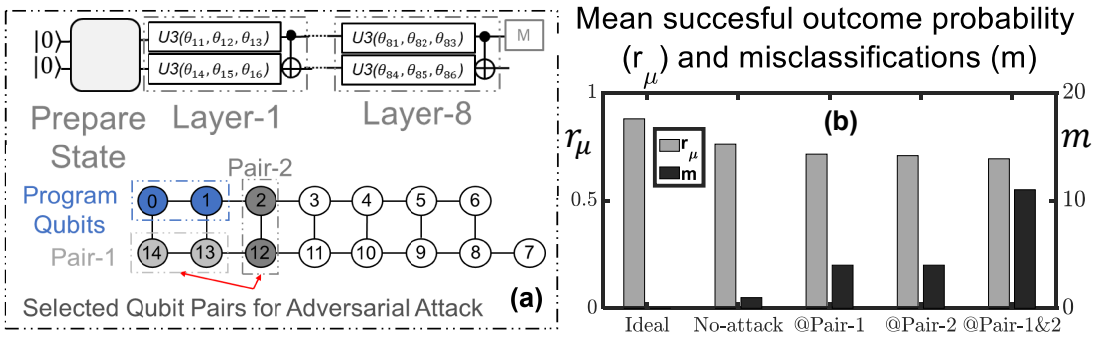


Figure 7. Impact of crosstalk on a 2-qubit quantum classifier. The experimental results show that crosstalk can lead to a higher number of misclassification.

Such crosstalk-induced fault injection attacks can be mitigated by introducing isolation/ buffer qubits between user programs [59]. If the user–1 program is allocated to {Q0,Q1} (Fig. 7), then another user program will not be allowed to use the neighboring qubits (i.e., {Q2, Q12, Q13, Q14}) of the user–1 program.

D. Robustness of QNN against Spatial Variations in Qubits

Error rates may vary among qubits in the same piece of hardware. This complicates optimal qubit selection for both training and inference of QNN models on hardware. Selecting a sub-optimal set of qubits can result in inefficient training and poor inference performance due to higher accumulation of error in the QNN circuits. The impact of qubit-to-qubit variations has been studied extensively [60]–[62]. Numerous generic strategies exist for exploiting variation-awareness in order to boost the performance of quantum programs running on hardware. The key idea is to assign the bulk of gates to less erroneous qubits. Many of these methods can be applied to QNN circuits directly.

In [60], the authors first proposed leveraging qubit-to-qubit variation to improve the program success rate. They proposed variation-aware qubit allocation (VQA) and variation-aware qubit movement (VQM) policies. In VQA, a set of physical qubits are picked to maximize their cumulative connectivity strength. The cumulative coupling strength is defined as the sum of success probabilities of all coupling links between the qubit and its neighbors. Coupling strength reflects two things: (i) a qubit is connected to more neighbors which is beneficial for optimal routing (less SWAP), and (b) the 2-qubit operations between the qubit and its neighbors will be less erroneous. Additionally, the VQM policy ensures that the compiler choose a routing path with fewer erroneous links. VQA can be used to determine the ideal set of qubits for hardware-based QNN training. VQM can help reducing error accumulation in QNN circuits caused by additional SWAP operations.

In [61], the authors started with a depth optimal NN-compliant version of the circuit using an algorithms as in [63] and searched for an isomorphic sub-graph from the device coupling graph with best program fidelity (QURE). The method contained the depth of the circuit (beneficial to counteract qubit lifetime issue) while finding better qubits and links to run the program. QURE is capable of determining an ideal set of qubits for hardware-based QNN inference.

In [62], the authors used satisfiability-modulo-theorem (SMT) to make qubit allocation and movement decisions while keeping error rate variations in mind. They also included readout error in their allocation decision besides gate error. The goal is to keep the circuit's overall error accumulation to a minimum. At the penalty of increased runtime, the reported performance improvement is substantially higher than the above heuristics. Identical circuit layers make up QNN circuits. One layer can be optimized and then used for the other layers. As a result, compared to other generic quantum programs, employing SMT to optimize QNN circuits is both appealing and effective, as well as less time consuming.

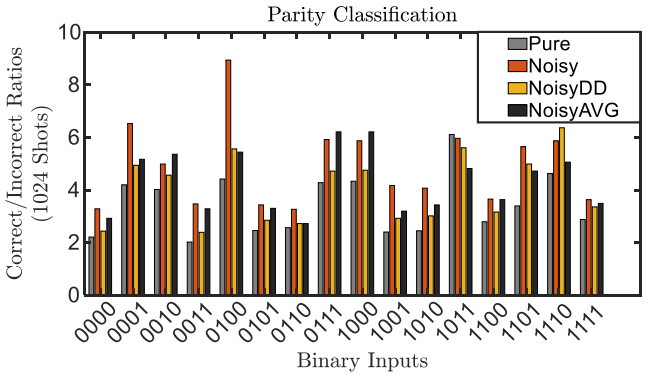


Figure 8. Inference performance of a 4-bit parity classifier (trained with three different approaches) on IBMQX4 [64].

E. Robustness of QNN against Temporal Variations in Qubits

The quality of qubits in terms of coherence times, gate error rates, measurement error rates, etc. may vary over time [54]. When the same quantum circuit is run on the same hardware at different times, it may produce variable results. This is especially concerning for QML models, as temporal changes can jeopardize their reliability.

The training of the quantum classifier leverages a quantum-classical hybrid loop where a PQC generates an output distribution, and a classical optimizer updates the parameters of the PQC based on the output to minimize a cost function (minimize loss during the training phase). In [64], we noted that if the training is performed including noise in the PQC, the quantum classifier shows more resilient performance. The classification accuracy for a quantum parity classifier from [64] is shown in Fig. 8. Here, "pure" is noiseless training, and "noisy" is noisy training with noise values of the respective day. The plot clearly shows training with noise has better accuracy compared to training without noise. However, noise values change over time. Therefore, the same trained classifier performs worse on a different day ("noisyDD"). As training a classifier every day is expensive, a reasonable trade-off is to use an average value for noise data collected over some time ("noisyAVG").

The work in [65] propose just-in-time compilation of quantum circuits to address temporal variability. The hardware quality can be assessed with standard calibration protocols before target program execution. The calibration data can be used by the circuit compiler to map the program to the best available qubits at the current hardware noise levels. Experiments indicate that the accuracy of circuit results improves by 3-304% with on-the-fly circuit mappings based on error measurements just prior to application execution. Just-in-time compilation can boost the performance of QNN models during inference on actual hardware.

F. Noise Mitigation Techniques for QNN

The possibility of QNN to attain quantum advantage on near-term Noisy Intermediate Scale Quantum (NISQ) computers is piquing researchers' interest. However, the performance of QNN models on real quantum devices suffers greatly due to high quantum noise. Techniques are proposed to mitigate these noises/errors e.g., multiple measurements of a quantum circuit are performed at different error rates and the ideal measurement results are extrapolated for noiseless case [66]. However, these techniques are generic, fail to take the unique characteristics of QNN into account and can only be applied to the QNN inference stage.

The work in [67] proposes a noise mitigation framework which optimizes QNN robustness in both training and inference stages. They use three main techniques to accomplish the same: (a) *Post-measurement normalization*, in which they match the distribution of measurement of the noise-free simulation with the real hardware, (b) *Quantum noise injection*, in which they insert error gates (in PQC and after measurement) based on a realistic noise model into the training process to increase the classification margin between classes. and (c) *Post-measurement quantization*, in which they quantize the measurement output to discrete values in order to further reduce noise. This framework leads to a significant increase in classification accuracy by up to 43% when measured on real quantum machines. As a result, this work significantly reduces the impact of quantum noise on QNN, opening the door to further QML applications.

In [68], the authors propose a methodology for efficient and scalable QNN training and inference on real quantum hardwares. First they employ the parameter shift rule, which states that the gradient of each parameter can be computed simply by shifting the parameter twice and calculating the difference between the two outputs to compute the quantum gradient directly on real quantum devices. Following gradient computation, they probabilistically prune the gradients with small magnitudes based on their distribution because noise would easily overwhelm these signals. Thus, removing the unreliable gradients improves both the model's reliability and performance, as it helps training converge faster by skipping the evaluation of those gradients. This method achieved accuracies roughly equivalent to noise-free simulations performed on classical computers while providing better training scalability.

VI. Conclusion

Considering the application of Deep Neural Networks in safety-critical applications, enhancing their dependability against unintentional faults as well as malicious fault attacks is essential. Accordingly, in this paper, we discuss the state-of-the-art schemes that can be deployed for improving the reliability of Deep Learning (DL) accelerators.

This paper presented a low-cost self-test scheme for DNN hardware along with a framework that enables the designers to assess the reliability of these architectures. Moreover, a training algorithm to diminish perturbations during inference time without highly scarifying the accuracy was proposed. In addition, we presented the schemes that can be used in Quantum Neural Networks to enhance their reliability against fault injections and spatial and temporal variations in Qubits.

REFERENCES

[1] L. M. Zhang, "Genetic deep neural networks using different activation functions for financial data mining," in *2015 IEEE International Conference on Big Data (Big Data)*, 2015, pp. 2849–2851.

[2] S. S. Sengar, U. Hariharan, and K. Rajkumar, "Multimodal biometric authentication system using deep learning method," in *2020 International Conference on Emerging Smart Computing and Informatics (ESCI)*, 2020, pp. 309–312.

[3] J. Shi, X. Fan, J. Wu, J. Chen, and W. Chen, "Deepdiagnosis: Dnn-based diagnosis prediction from pediatric big healthcare data," in *2018 Sixth International Conference on Advanced Cloud and Big Data (CBD)*, 2018, pp. 287–292.

[4] F. Yu, Z. Qin, C. Liu, D. Wang, and X. Chen, "Rein the robuts: Robust dnn-based image recognition in autonomous driving systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 6, pp. 1258–1271, 2021.

[5] D. Strigl, K. Kofler, and S. Podlipnig, "Performance and scalability of gpu-based convolutional neural networks," in *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, 2010, pp. 317–324.

[6] X. Li, G. Zhang, H. H. Huang, Z. Wang, and W. Zheng, "Performance analysis of gpu-based convolutional neural networks," in *2016 45th International Conference on Parallel Processing (ICPP)*, 2016, pp. 67–76.

[7] H. Jang, A. Park, and K. Jung, "Neural network implementation using cuda and openmp," in *2008 Digital Image Computing: Techniques and Applications*, 2008, pp. 155–161.

[8] D. A. Padilla, R. A. I. Pajes, and J. T. De Guzman, "Detection of corn leaf diseases using convolutional neural network with openmp implementation," in *2020 IEEE 12th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment, and Management (HNICEM)*, 2020, pp. 1–6.

[9] E. Ozen and A. Orailoglu, "Sanity-check: Boosting the reliability of safety-critical deep neural network applications," in *2019 IEEE 28th Asian Test Symposium (ATS)*, 2019, pp. 7–75.

[10] ——, "Just say zero: Containing critical bit-error propagation in deep neural networks with anomalous feature suppression," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020, pp. 1–9.

[11] M. A. Neggaz, I. Alouani, P. R. Lorenzo, and S. Niar, "A reliability study on cnns for critical embedded systems," in *2018 IEEE 36th International Conference on Computer Design (ICCD)*, 2018, pp. 476–479.

[12] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey of machine learning accelerators," in *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, 2020, pp. 1–12.

[13] J. Zhang, K. Rangineni, Z. Ghodsi, and S. Garg, "Thundervolt: Enabling aggressive voltage underscaling and timing error resilience for energy efficient deep learning accelerators," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.

[14] W. Choi, D. Shin, J. Park, and S. Ghosh, "Sensitivity based error resilient techniques for energy efficient deep neural network accelerators," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[15] I. 26262, "Road vehicles-Functional safety." ISO, 2018, Accessed: 11-Oct-2020. [Online]. Available: https://www.iso.org/standard/68383.html

[16] B. Reagen *et al.*, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 267–278.

[17] G. Li *et al.*, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17, 2017, pp. 8:1–8:12.

[18] B. Reagen *et al.*, "Ares: A framework for quantifying the resilience of deep neural networks," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, June 2018, pp. 1–6.

[19] N. Chandramoorthy *et al.*, "Resilient low voltage accelerators for high energy efficiency," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 147–158.

[20] P. N. Whatmough *et al.*, "14.3 a 28nm soc with a 1.2ghz 568nj/prediction sparse deep-neural-network engine with >0.1 timing error rate tolerance for iot applications," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, Feb 2017, pp. 242–243.

[21] Y. He, P. Balaprakash, and Y. Li, "Fidelity: Efficient resilience analysis framework for deep learning accelerators," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 270–281.

[22] S. Mitra, P. Bose, E. Cheng, C.-Y. Cher, H. Cho, R. Joshi, Y. M. Kim, C. R. Lefurgy, Y. Li, K. P. Rodbell, K. Skadron, J. Stathis, and L. Szafaryn, "The resilience wall: Cross-layer solution strategies," in *Proceedings of Technical Program - 2014 International Symposium on VLSI Technology, Systems and Application (VLSI-TSA)*, 2014, pp. 1–11.

[23] Y. He, T. Uezono, and Y. Li, "Efficient functional in-field self-test for deep learning accelerators," in *2021 IEEE International Test Conference (ITC)*, 2021, pp. 93–102.

[24] Y. Li, S. Makar, and S. Mitra, "Casp: Concurrent autonomous chip self-test using stored test patterns," in *2008 Design, Automation and Test in Europe*, 2008, pp. 885–890.

[25] Y. Li, O. Mutlu, D. S. Gardner, and S. Mitra, "Concurrent autonomous self-test for uncore components in system-on-chips," in *2010 28th VLSI Test Symposium (VTS)*, 2010, pp. 232–237.

[26] P. Bardell, W. McAnney, and J. Savir, *Built In Test for VLSI: Pseudo-random Techniques*. Wiley, 1987.

[27] H. Cho *et al.*, "Quantitative evaluation of soft error injection techniques for robust system design," in *Proceedings of the 50th Annual Design Automation Conference*, 2013, pp. 1–10.

[28] E. Cheng, S. Mirkhani, L. G. Szafaryn, C.-Y. Cher, H. Cho, K. Skadron, M. R. Stan, K. Lilja, J. A. Abraham, P. Bose, and S. Mitra, "Clear: Cross-layer exploration for architecting resilience - combining hardware and software techniques to tolerate soft errors in processor cores," in *Proceedings of the 53rd Annual Design Automation Conference*, ser. DAC '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2897937.2897996

[29] M. Sadi and M. Tehranipoor, "Design of a network of digital sensor macros for extracting power supply noise profile in socs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 5, pp. 1702–1714, 2016.

[30] M. Sadi, G. K. Contreras, J. Chen, L. Winemberg, and M. Tehranipoor, "Design of reliable socs with bist hardware and machine learning," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 11, pp. 3237–3250, 2017.

[31] M. Sadi, G. K. Contreras, J. Chen, L. Winemberg, and M. Tehranipoor, "Design of Reliable SoCs With BIST Hardware and Machine Learning," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 11, pp. 3237–3250, 2017.

[32] G. L. et al., "Understanding Error Propagation in Deep Learning Neural Network (DNN) Accelerators and Applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17, New York, NY, USA, 2017.

[33] S. Kundu, K. Basu, M. Sadi, T. Titirsha, S. Song, A. Das, and U. Guin, "Special session: Reliability analysis for ai/ml hardware," in *2021 IEEE 39th VLSI Test Symposium (VTS)*, 2021, pp. 1–10.

[34] M. Sadi and U. Guin, "Test and yield loss reduction of ai and deep learning accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 1, pp. 104–115, 2022.

[35] K. Mishty and M. Sadi, "Designing efficient and high-performance ai accelerators with customized stt-mram," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 10, pp. 1730–1742, 2021.

[36] G. Tshagharyan, G. Harutyunyan, and Y. Zorian, "An effective functional safety solution for automotive systems-on-chip," in *2017 IEEE International Test Conference (ITC)*, 2017, pp. 1–10.

[37] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[38] P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson, "Averaging Weights Leads to Wider Optima and Better Generalization," 2019.

[39] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," in *Neural Information Processing Systems*, 2018.

[40] B. A. Pearlmutter, "Fast Exact Multiplication by the Hessian," *Neural Computation*, vol. 6, no. 1, pp. 147–160, 1994.

[41] Z. Yao, A. Gholami, S. Shen, K. Keutzer, and M. W. Mahoney, "Adahessian: An adaptive second order optimizer for machine learning," *Association for the Advancement of Artificial Intelligence (AAAI)*, 2021.

[42] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning," *Nature*, vol. 549, no. 7671, pp. 195–202, 2017.
[43] N. Killoran *et al.*, "Continuous-variable quantum neural networks," *Physical Review Research*, vol. 1, no. 3, p. 033063, 2019.
[44] P.-L. Dallaire-Demers and N. Killoran, "Quantum generative adversarial networks," *Physical Review A*, vol. 98, no. 1, p. 012324, 2018.
[45] M. Schuld, A. Bocharov, K. M. Svore, and N. Wiebe, "Circuit-centric quantum classifiers," *Physical Review A*, vol. 101, no. 3, p. 032308, 2020.
[46] E. Farhi and H. Neven, "Classification with quantum neural networks on near term processors," *arXiv preprint arXiv:1802.06002*, 2018.
[47] M. Schuld *et al.*, "Effect of data encoding on the expressive power of variational quantum-machine-learning models," *Physical Review A*, 2021.
[48] S. Lloyd *et al.*, "Quantum embeddings for machine learning," *arXiv preprint arXiv:2001.03622*, 2020.
[49] A. Abbas *et al.*, "The power of quantum neural networks," *Nature Computational Science*, vol. 1, no. 6, pp. 403–409, 2021.
[50] Y. Du *et al.*, "Expressive power of parametrized quantum circuits," *Physical Review Research*, vol. 2, no. 3, p. 033125, 2020.
[51] J. R. McClean, S. Boixo, V. N. Smelyanskiy, R. Babbush, and H. Neven, "Barren plateaus in quantum neural network training landscapes," *Nature communications*, vol. 9, no. 1, pp. 1–6, 2018.
[52] S. Wang, E. Fontana, M. Cerezo, K. Sharma, A. Sone, L. Cincio, and P. J. Coles, "Noise-induced barren plateaus in variational quantum algorithms," *arXiv preprint arXiv:2007.14384*, 2020.
[53] S. S. Tannu and M. K. Qureshi, "Not all qubits are created equal: a case for variability-aware policies for nisq-era quantum computers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 987–999.
[54] M. Alam *et al.*, "Addressing temporal variations in qubit quality metrics for parameterized quantum circuits," in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2019, pp. 1–6.
[55] N. Liu and P. Wittek, "Vulnerability of quantum classification to adversarial perturbations," *Physical Review A*, vol. 101, no. 6, p. 062331, 2020.
[56] A. A. Saki, M. Alam, and S. Ghosh, "Analysis of crosstalk in NISQ devices and security implications in multi-programming regime," in *2020 IEEE/ACM ISLPED*. IEEE, 2020, pp. 1–6.
[57] S. Lu, L.-M. Duan, and D.-L. Deng, "Quantum adversarial machine learning," *Physical Review Research*, vol. 2, no. 3, p. 033212, 2020.
[58] S. Sim *et al.*, "Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms," *Advanced Quantum Technologies*, vol. 2, no. 12, p. 1900070, 2019.
[59] A. Ash-Saki, M. Alam, and S. Ghosh, "Analysis of crosstalk in NISQ devices and security implications in multi-programming regime," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, 2020, pp. 25–30.
[60] S. S. Tannu and M. K. Qureshi, "Not all qubits are created equal: a case for variability-aware policies for NISQ-era quantum computers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 987–999.
[61] A. Ash-Saki, M. Alam, and S. Ghosh, "QURE: Qubit re-allocation in noisy intermediate-scale quantum computers," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.
[62] P. Murali, J. M. Baker, A. Javadi-Abhari, F. T. Chong, and M. Martonosi, "Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 1015–1029.
[63] A. Zulehner, A. Paler, and R. Wille, "An efficient methodology for mapping quantum circuits to the IBM QX architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 7, pp. 1226–1236, 2018.
[64] M. Alam, A. Ash-Saki, and S. Ghosh, "Addressing Temporal Variations in Qubit Quality Metrics for Parameterized Quantum Circuits," in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2019, pp. 1–6.
[65] E. Wilson, S. Singh, and F. Mueller, "Just-in-time quantum circuit transpilation reduces noise," in *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2020, pp. 345–355.
[66] K. Temme, S. Bravyi, and J. M. Gambetta, "Error mitigation for short-depth quantum circuits," *Phys. Rev. Lett.*, vol. 119, p. 180509, Nov 2017. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.119.180509
[67] H. Wang, J. Gu, Y. Ding, Z. Li, F. T. Chong, D. Z. Pan, and S. Han, "Roqnn: Noise-aware training for robust quantum neural networks," *arXiv preprint arXiv:2110.11331*, 2021.
[68] H. Wang, Z. Li, J. Gu, Y. Ding, D. Z. Pan, and S. Han, "On-chip qnn: Towards efficient on-chip training of quantum neural networks," *arXiv preprint arXiv:2202.13239*, 2022.

View publication stats