EasyChair Preprint

№ 15357

Trustworthiness Evaluation of Deep Learning Accelerators Using UVM-based Verification with Error Injection

Randa Aboudeif, Tasneem A. Awaad, Mohamed Abdelsalam and Yehea Ismail

November 3, 2024

Trustworthiness Evaluation of Deep Learning Accelerators Using UVM-based Verification with Error Injection

Randa Aboudeif*†, Tasneem A. Awaad ‡, Mohamed AbdElsalam†, Yehea Ismail*
*Electronics and Communications Engineering Department, The American University in Cairo, Cairo, Egypt
{aranda, Y.ismail}@aucegypt.edu
†Siemens Digital Industries Software, Cairo, Egypt
{randa.aboudief, mohamed.abd_el_salam_ahmed}@siemens.com
‡ Department of Computer and Systems Engineering, Faculty of Engineering, Ain Shams University, Cairo, Egypt
tasneem.awaad@eng.asu.edu.eg

***Abstract*—Testing the reliability and trustworthiness of high-performance computing (HPC) applications has made Deep Learning Accelerators (DLAs) verification critically important. In this paper, we introduce a hardware verification framework with an error injection methodology based on the Universal Verification Methodology (UVM) for DLAs that is scalable, reusable, and efficient to test the robustness and resilience of Deep Neural Networks (DNNs) running on various DLA designs. Furthermore, the error injection methodology is applicable to simulation and hardware-assisted verification (HAV) platforms for emulation and FPGA prototyping. Our proposed error injection mechanism is evaluated using Nvidia Deep Learning Accelerator (NVDLA), an open-source DLA core.**

***Index Terms*—CNN, UVM, Deep Learning Accelerators, verification, Error injection, NVDLA**

I. Introduction

High-performance computing applications, including speech recognition, computer vision, and image classification have turned to hardware accelerators, known as Deep Learning Accelerators, as a preferred solution because of recent deep learning (DL) advances. Moreover, the next generation of HPC Electronics Control Unit (ECU) designs requires adding specialized DLAs for their low power consumption and resource efficiency. DNNs demonstrated outstanding performance in Learning Enabled Autonomous Systems (LEAS) for essential autonomous driving functions in these ECUs, like perception.

DLAs consist of hundreds of parallel processing engines and can access pre-trained networks through on-chip memory or the cloud. Because DLAs are involved in safety-critical applications, their reliability is critical for assessment. This is especially important with the noticeable increase in sensor faults, adversarial attacks, and hardware functional errors occurring in DLAs resulting in violations of safety and reliability requirements. [1].

Identifying and resolving design issues in DLAs resulting from the massive parallel multiplications and accumulation (MAC) operations involved in each DNN layer mapped to hardware has been the main target of any DLA verification process. These challenges are addressed in [2] with a proposed DLA verification methodology, using UVM concepts. A robust self-checking verification methodology for error injection and detection is needed to tackle these DLA design challenges. This methodology should mimic faulty data input and functional error conditions, identify errors that can be injected into the DLA design, classify them, and validate the DLA design response.

UVM is an industry standard that is used in the verification of digital hardware, developed by Accellera [3]. Verification components used by UVM are generic, object-oriented, reusable, and scalable. UVM supports functional coverage and constrained random stimulus generation capabilities used to test the design under test (DUT). Furthermore, simulation for UVM-based verification provides more capability and modeling flexibility across different design verification stages at the chip, subsystem, and entire system level. In addition to that, emulation and FPGA prototyping achieve significant performance benefits in the functional verification flow by speeding up complex test cases to run faster and hit more corner cases. Therefore, DLA verification requires hardware acceleration using emulation platforms and FPGA prototyping to improve the verification process performance.

In this paper, the work in [2] is extended with a new error injection mechanism, which is now part of that UVM-based DLA verification framework. The mechanism is scalable and can be applied to each convolutional neural network (CNN) layer in any CNN to test the resilience and ability of each layer to mitigate the effect of an injected error and thus test the trustworthiness of different DLA designs. We showcase this new feature in simulation and emulation. We used NVDLA from Nvidia Xavier SoC [4], which is an open architecture hardware accelerator, as an example of DLA design under verification to prove the robustness of our proposed verification methodology with an error injection mechanism.

The remainder of the paper is organized as follows: Section II discusses the background. Section III specifies the proposed methodology. Section IV illustrates the experimental results. Finally, conclusions and future work are discussed in Section

V.

II. Background

In this section, we briefly illustrate the concept of hardware faults and demonstrate how fault injection can be performed on deep neural networks.

A. Hardware faults

Hardware faults are categorized based on fault duration into permanent and transient faults. Until corrective action is performed, a permanent fault caused by physical defects in the hardware remains active. A transient fault is only active for a short time. When a transient fault happens in a system, it might corrupt the output of an application or cause the system to crash. Furthermore, as many DNN systems are safety-critical, soft errors can have catastrophic consequences, and error mitigation is necessary for such systems to achieve specific reliability goals. Similar to self-driving vehicles, where a minor error can result in the misclassification of an object, leading to incorrect actions by the vehicle. For example, a truck can be misclassified under soft error. In the case of no errors, the coming object should be classified as a transport truck by the DNN in the car, and the brakes should be applied in time to avoid a collision. Nevertheless, if a soft error occurs in the DNN, then the truck may be erroneously classified as a bird. This is a serious concern since it leads to violating standards like ISO 26262, which addresses the vehicles' functional safety [5], and ISO 21448 [6] which outlines limitations in achieving the intended functionality of Machine Learning components.

B. Fault Injection for Deep Neural Networks

To implement an efficient fault injection methodology, some concerns must be taken into consideration. This is because measuring the effect of the fault must be done differently from measuring the typical fault injection.

Numerous fault injection frameworks have been proposed in the literature. For example:

- In [7], for convolutional layers in systolic array RTL DNN accelerators at channel granularity, a cross-layer fault injection methodology is proposed by simulating the RTL Channel Under Test (ChUT) execution. Then, to determine the impact of the injected faults at the application level, the DNN execution is completed using the faulty outputs from the RTL simulation at the software level. This framework runs faster in terms of the fault injection time and has parallel capabilities.
- A methodology for the reliability analysis of CNNs is presented in [8], using an error simulation engine and validated error models taken from an extensive fault injection approach. This framework is easy to use, controllable, flexible, and fast while combining the accuracy of architecture-level and application-level fault injection. In terms of speed and the ability to reproduce the effects of faults on the final CNN output, this methodology achieved higher accuracy than the TensorFI (Software-implemented Fault Injection tool) [9] functional error simulator [10] and the innovative SASSIFI fault injection environment [11] for CNNs accelerated onto GPU.
- In [1], a canonical model of the DNN accelerator hardware is used to modify a DNN simulator and give a fault injection technique for four popular neural networks for image recognition. Furthermore, a large-scale study on fault injection is conducted for the faults occurring in the accelerators' data path. Then, depending on the neural network's architecture, data types, layers' positions, and types, the error propagation behaviors are categorized.

III. Proposed Methodology

The proposed methodology is to implement a scalable error injection mechanism for testing the trustworthiness of deep learning accelerators with simulation, emulation, and FPGA prototyping to detect errors at the early stages of the DLA verification process. The proposed methodology applies error injection using three mechanisms. Firstly, it applies error injection in the DNN data path by corrupting the DNN layers; feature map, weight, and bias for each DNN layer. Secondly, it applies adversarial attacks on the input image to mitigate the perturbation in input received in the physical world from cameras and other sensors. Thirdly, it applies error injection in the DLA hardware configurations to detect faulty hardware configurations for a DNN that may disrupt the inference process. The proposed error injection mechanism is scalable and reliable. It provides large coverage, can hit corner cases, and has complete access to the DNN data path between layers and the DLA configurations. To the best of our knowledge, this is the first contribution to implementing an error injection mechanism for verifying DLA robustness using UVM. All the related work is based on direct testing by applying error injection for a specific DLA design without using UVM or applying error injection for a specific DNN architecture. The proposed error injection methodology did not add any extra runtime or performance overhead compared to other fault injection mechanisms. Furthermore, the proposed framework can be portable across the various HAV platforms to accelerate the verification process.

As mentioned in the introduction, the proposed error injection methodology is an extension to the framework for DLA verification in [2]. This is a UVM-based testbench established for testing CNN inference on the Deep Learning Accelerators for simulation and emulation on HAV platforms, and now with error injection methodology as shown in Fig. 1. In SoC verification, hardware acceleration has become important. Emulation can speed up complex test cases so they run faster and catch corner case errors. Testbench-Xpress (TBX) technology is used in the proposed UVM environment with error injection to support both hardware emulation and simulation. TBX is compliant with the Standard Co-Emulation Modeling Interface (SCE-MI), which is a transaction-level communication channel between the DLA DUT mapped on the hardware emulation

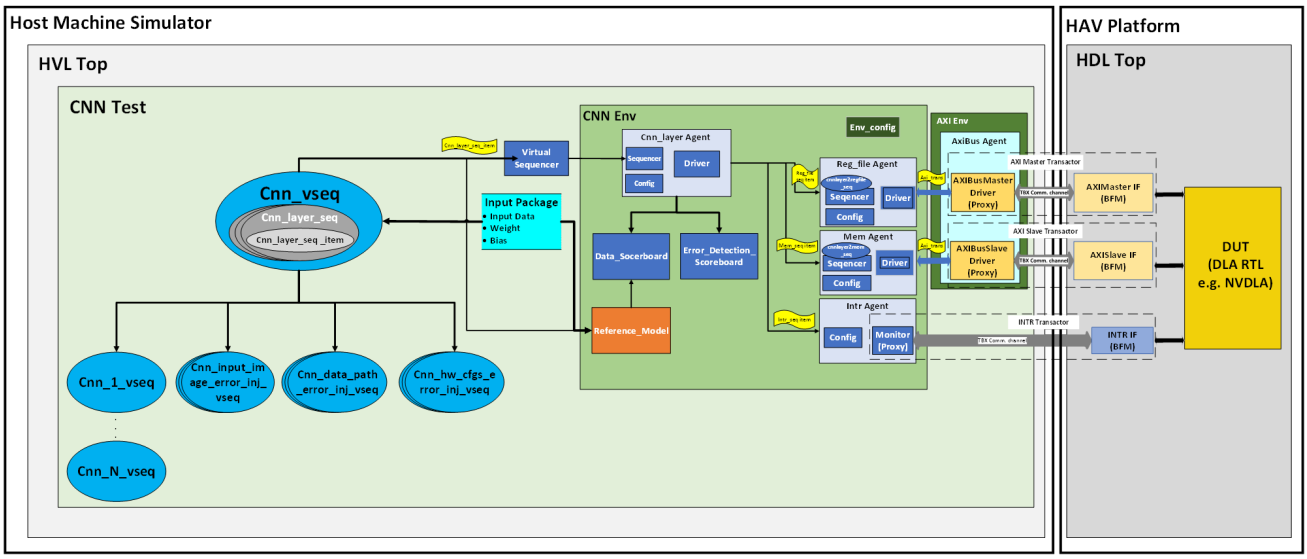

Fig. 1: UVM testbench architecture with the added error injection methodology.

platform and the host machine's UVM testbench verification environment, which is compatible with TBX [12].

Therefore, the architecture of the proposed UVM environment, which is divided into Hardware Verification Language (HVL) Top and Hardware Description Language (HDL) Top, is based on the dual top concept. The HVL Top is class-based, dynamic, behavioral, and untimed. Thus, it is responsible to execute the UVM test, which is the testbench part that is running on the host machine. While the HDL Top has the DUT instance and its connection to the UVM testbench virtual interfaces, which is the hardware part that is running on the emulator (HAV platform), the HDL Top is timed and synthesizable [13].

A. DNN Data Path Error Injection

Error injection for the DNN data path is done in this framework by extending a new sequence for error injection from the cnn_vseq, which is a UVM virtual base sequence. This virtual sequence's responsibility is to run the cnn_layer_seq, which is used for running a CNN on the DLA. cnn_layer_seq sequence constructs the cnn_layer_seq_item transactions that include the DLA hardware configurations, data, weight, and bias specific for each CNN layer. Then these transactions are transmitted using the virtual sequencer to the cnn_layer_agent's driver, as shown in Fig. 1. After that, the cnn_layer_agent's driver splits down these received transactions to a lower level of abstraction to be transmitted to the DLA DUT in terms of registers and memory read/write operations. Firstly, the Reg_file seq_item transaction is used to transmit the DLA configuration through the Reg_file Agent's driver in the form of register read/write operations that are sent to the DLA DUT AXI interface from the AXIBusMaster Driver. The AXIBusMaster Driver is part of the AXIBus Agent in the AXI environment. Secondly, the mem_seq_item transaction is used to transmit the CNN layers' data, weight, and bias through the mem agent in the form of memory read/write operations that are sent to the DLA DUT DRAM AXI interface from the AXIBusSlave Driver. The AXIBusSlave Driver is also part of the AXIBus Agent in the AXI environment. In addition to that, the interrupt received from the DLA DUT is handled through the intr agent using the intr_seq_item and then sent to the cnn_layer_seq through the CNN layer agent's driver.

Furthermore, the error injection for the CNN internal layers' input data, weight, and bias is done by randomly injecting errors for single and multiple values. This is done by corrupting a single memory location in the case of a single value and corrupting randomly chosen more than one memory location in the case of multiple value error injection. Moreover, this error injection mechanism is done at different positions in the CNN (at different layers) to study the error propagation in the CNN and the resilience of each layer. Furthermore, to investigate the effect of the error injection in each aspect of the DLA inference process in terms of CNN robustness and output predictions. Table I lists the proposed data path error injection testing scenarios that are applicable for any CNN and scalable for single and multiple-layer convolutional neural networks.

B. DNN Input image Error Injection

The error injection is done on the input images in the proposed methodology by generating adversarial examples from the original input data to test the robustness of the CNN running on the DLA DUT. An adversarial example is a sample of input data that has been slightly modified to make a CNN misclassify it intentionally. CNN still fails, even if these changes are so small that a human observer would not even notice them. Adversarial examples cause security concerns because they could be used to attack deep learning systems, even if the adversary cannot access the underlying model [14]. In addition to that, the input image error injection mechanism

TABLE I: Data path error injection testing scenarios.

Testing scenarios	Features to test
1. Single value feature map error injection	Single random incorrect data value injected in a randomly chosen single or multiple CNN layers.
2. Multiple values feature map error injection	Multiple random incorrect data values injected in a randomly chosen single or multiple CNN layers.
3. Single value weight error injection	Single random incorrect weight value injected in a randomly chosen single or multiple CNN layers.
4. Multiple values weight error injection	Multiple random incorrect weight values injected in a randomly chosen single or multiple CNN layers.
5. Single value bias error injection	Single random incorrect bias value injected in a randomly chosen single or multiple CNN layers.
6. Multiple values bias error injection	Multiple random incorrect bias values injected in a randomly chosen single or multiple CNN layers.

helps in detecting hardware faults that may happen in the DLA system memory and affect the input data memory locations. Which consequently negatively affects the inference process.

The fast gradient sign method (FGSM) [14] attack is used in the proposed error injection mechanism to generate the adversarial examples. A Python script is used to implement this method on the input image during the data preparation stage illustrated in [2] to generate the adversarial image to be used during the inference process as shown in Fig. 2. The fast gradient sign method uses the neural network's gradients to generate an adversarial example. The method takes an input image and creates a new image that maximizes the loss using the loss gradients with respect to the input image. We refer to this newly created image as the adversarial image. The method can be explained using the following equation [15]:

$$\mathbf{adv\_img} = \mathbf{in\_img} + \epsilon \cdot \text{sign}(\nabla_{\mathbf{in\_img}} J(\theta, \mathbf{in\_img}, in\_lbl)),$$

where:

- adv_img: Adversarial image
- in_img: Original input image.
- in_lbl: Original input label.
- ϵ: A perturbation factor.
- θ : Model parameters.
- $\nabla_{\mathbf{in\_img}} J(\theta, \mathbf{in\_img}, in\_lbl)$: The gradient of the loss function $J(\theta, \mathbf{in\_img}, in\_lbl)$ with respect to the input $\mathbf{in\_img}$.

This approach is quite fast because it is simple to apply the chain rule to calculate the required gradients and identify how each input pixel affects the loss. Hence, the goal is to deceive an already-trained model using these adversarial-generated images. Therefore, the cnn_vseq base sequence is extended to implement a new sequence to run the CNN on the DLA using the generated adversarial images as shown in Fig. 1.

C. DLA hardware registers Error Injection

Error injection for the DLA hardware configurations register space is done in this framework by extending a sequence from the cnn_vseq for DLA configurations error-injecting to randomly corrupt the hardware configurations of the DLA for a specific CNN (included in the cnn_layer_seq_item), such as changing the memory address configurations for data, weight, and bias, changing the feature map parameters for different CNN layers, or the weight kernels' parameters. In addition to that, the error detection scoreboard shown in Fig. 1 is implemented to check and report the DLA behavior in case of incorrect or unexpected hardware configuration during the inference process for a CNN.

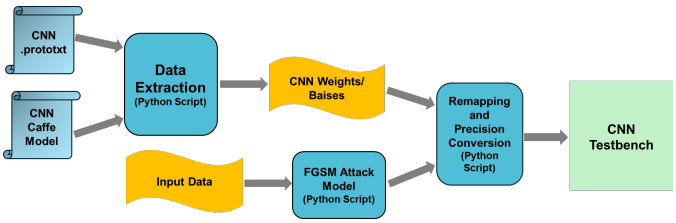

Fig. 2: Inupt data and weight pre-processing with input image error injection.

IV. EXPERIMENTAL RESULTS

A case study is done on the NVDLA, as the proposed methodology is used to verify the NVDLA inference function and test the running CNN resilience for simulation and emulation with different error injection testing scenarios. The NVDLA DUT is integrated with the UVM testbench through AXI-based interfaces, as demonstrated in [2]. Each CNN testing scenario configures the NVDLA blocks required for each CNN layer according to the layer's parameters and then sends the required input data, pre-trained weight, and bias to the NVDLA through its DRAM interface.

The proposed error injection testing scenarios are applicable to sophisticated custom and standard CNN architectures as per the CNN sequence for programming the NVDLA core. As a showcase, two error injection testing scenarios are implemented for a single CNN convolution layer with inference parameters shown in Table II and for the LeNet-5 CNN. LeNet-5 CNN was proposed by LeCun in 1998 [16], which was successfully applied to handwritten digit recognition. LeNet-5 consists of the following layers: an input layer, two convolution layers each followed by a pooling layer, two fully connected layers, and an output layer.

TABLE II: The single CNN convolution layer parameters.

Layer	Input	filters No.	Filter size	Stride	Output
Convolution	8*8*32	16	3*3*32	1	1x1x16

The architecture and the parameters used for inference are mentioned in Table III [17]. The used dataset for testing is the MNIST handwritten digit dataset. QuestaSim simulator tool is used for simulation [18]. Moreover, the Veloce Strato platform is used for emulation [19].

A. Data Path Error Injection Results

Error injection testing scenarios are implemented for running the single CNN convolution layer on the NVDLA by

TABLE III: The LeNet-5 CNN parameters.

Layer	Input	Filter	Stride	Output
Convolution 1	28*28*1	5*5*1*20	1	24*24*20
Pooling 1	24*24*20	2*2	2	12*12*20
Convolution 2	12*12*20	5*5*20*50	1	8*8*50
Pooling 2	8*8*50	2*2	2	4*4*50
Fully Connected 1	4*4*50	4*4*50*50	1	1*1*50
Fully Connected 2	1*1*50	1*1*50*10	1	1*1*10

randomly injecting errors in weight, and the layer's input data (feature map) during the inference process for both simulation and emulation. The simulation and emulation regression results shown in Fig. 3 indicate that the convolution layer is more sensitive to multiple data errors than the multiple weight errors. Moreover, single data and single weight errors are almost masked in the convolution layer due to the presence of the ReLU activation function.

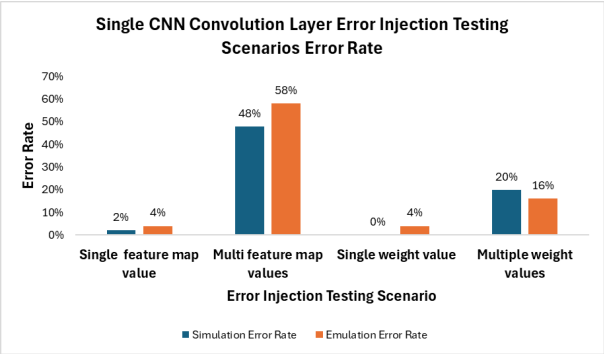

Fig. 3: Single CNN convolution layer data path error injection testing scenarios error rate.

Furthermore, error injection testing scenarios are implemented for LeNet-5 CNN running on the NVDLA through randomly injecting errors in the different layers' weight, bias, and the internal layers' input data (feature maps between layers) during the inference process for both simulation and emulation. The error injection is done by inserting random and incorrect single or multiple values of weight, bias, or internal layers' input data for a randomly chosen layer. Simulation and emulation regression are run for the implemented error injection testing scenarios over a random MINST dataset testing samples. As demonstrated in Fig. 4, the results indicate that in the LeNet-5 CNN, the majority of single-value input data errors within internal layers are masked and do not impact the output predictions of the final layer. Moreover, the LeNet-5 CNN is sensitive to the multiple values in internal layers' input data errors. However, some of them are masked by the POOL layers if they were injected into the convolution layers. Furthermore, the LeNet-5 CNN masks most of the single-value errors in weight. However, some errors in the second convolution layer were propagated and corrupted the last layer output predictions. Moreover, the LeNet-5 CNN is sensitive to multiple weight value errors, mainly if injected in the convolution layers. For the bias, the LeNet-5 CNN masks most of the single-value errors which reduced the CNN error rate. However, the LeNet-5 CNN is sensitive to the multiple-value bias errors and propagates them as they corrupt the output predictions except for those injected in the fully connected layers. In summary, the LeNet-5 CNN is sensitive to the internal layers' multiple values of input data corruption and weight corruption more than that of the bias corruption and less sensitive to single values corruption in data, weight, and bias propagation between layers.

B. Input image Error Injection Results

A testing scenario is implemented for running LeNet-5 CNN on the NVDLA using adversarial images as input images for simulation and emulation. Fig. 5 shows that the LeNet-5 CNN accuracy rate decreases with high values of ϵ as the fast gradient sign method adds noise scaled by ϵ to the image.

To sum up, as shown in Fig. 4, Fig. 3, and Fig. 5, some of the error injection testing scenarios running on the emulator have higher error rates compared to those running for simulation. This is because emulation tends to hit more bugs compared to simulation as it executes the design in a more realistic and faster environment, tests more scenarios, and operates at a lower level of abstraction. This leads to a higher likelihood of uncovering bugs that might not be exposed during simulation. However, emulation is useful in detecting more bugs, but simulation also often provides better tools for visibility and debugging.

TABLE IV: Error injection simulation and emulation runtime.

CNN	Simulation runtime	Total emulation runtime
Single CNN convolution layer with error injection	20.48 us	3.12s
LeNet-5 CNN with error injection test case	0.144 ms	19.42s

The proposed error injection methodology is faster compared to other introduced frameworks that are based on software instruction-by-instruction execution because the proposed UVM framework is a native code execution that runs very fast on a host machine. Table IV shows the simulation and emulation runtime for both the single CNN convolution layer and the LeNet-5 error injection scenarios, these runtime values are similar to those for the same testing scenarios without the error injection mechanism mentioned in [2]. The proposed error injection testing scenarios did not consume any extra runtime in simulation or emulation for the applicable running CNN on the DLA. Therefore, there is no overhead in the runtime while running the proposed testbench with the error injection testing scenarios. Moreover, this framework has a larger coverage for running any CNN, hitting corner cases, and is faster and easier to debug compared to other error injection mechanisms. The proposed framework added more visibility during the NVDLA testing and debugging, allowing direct and full access to the NVDLA register space for configuration with less runtime.

V. Conclusion

This work proposes our error injection methodology as part of the UVM-based verification framework for verifying

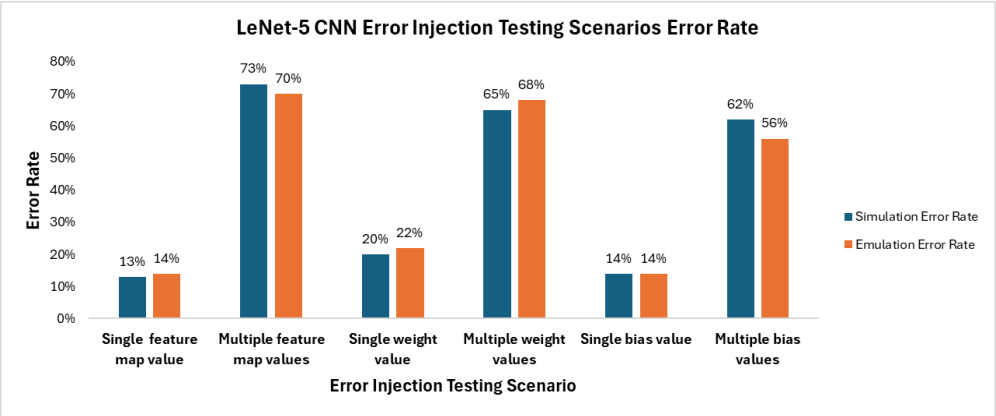

Fig. 4: LeNet-5 CNN data path error injection testing scenarios error rate.

deep learning accelerators' inference function using a generic and scalable UVM environment to run CNNs with different inference parameters in simulation and emulation. The illustrated error injection methodology in this paper has less simulation and emulation runtime to test the trustworthiness of complex DLA designs in each CNN Layer mapped to hardware, mainly in the presence of data corruption either due to hardware faults or input perturbations. Furthermore, the proposed methodology added more flexibility and scalability as it introduces cross-layer error injection in the DNN. As a future work, the proposed methodology will be applied to analyze the resilience of more complex CNNs with and without defense mechanisms running on the NVDLA against faults and attacks.

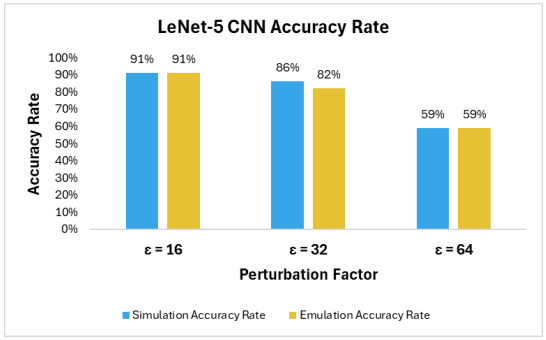

Fig. 5: LeNet-5 CNN with adversarial images accuracy rate.

REFERENCES

[1] G. Li et al., "Understanding Error Propagation in Deep Learning Neural Network (DNN) Accelerators and Applications," SC17: International Conference for High Performance Computing, Networking, Storage and Analysis, Denver, CO, USA, 2017, pp. 1-12.

[2] R. Aboudeif, T. A. Awaad, M. AbdelElSalam, and Y. Ismail "UVM Based Verification Framework for Deep Learning Hardware Accelerator: Case Study" International Conference on Electrical, Computer and Energy Technologies 2024; Sydney, Australia.

[3] Universal Verification Methodology (UVM) 1.2 User's Guide, 2015. https://www.accellera.org/images/downloads/standard /uvm/uvm_users_guide_1.2.pdf (accessed Sep. 12, 2024).

[4] "Nvidia Jetson xavier series," NVIDIA, https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier (accessed Sep. 12, 2024).

[5] ISO. 2018. ISO 26262: 2018 - Road vehicles – Functional safety. Standard. International Organization for Standardization.

[6] ISO. 2019. ISO/PAS 21448: 2019 - Road vehicles — Safety of the intended functionality. Standard. International Organization for Standardization.

[7] S. Pappalardo, A. Ruospo, I. O'Connor, B. Deveautour, E. Sanchez and A. Bosio, "A Fault Injection Framework for AI Hardware Accelerators," 2023 IEEE 24th Latin American Test Symposium (LATS), Veracruz, Mexico, 2023, pp. 1-6, doi: 10.1109/LATS58125.2023.10154505.

[8] C. Bolchini, L. Cassano, A. Miele, and A. Toschi, "Fast and Accurate Error Simulation for CNNs against Soft Errors," IEEE Transactions on Computers, vol. 72, no. 4, pp. 984–997, Apr. 2023, doi: https://doi.org/10.1109/TC.2022.3184274.

[9] Z. Chen, N. Narayanan, B. Fang, G. Li, K. Pattabiraman, and N. DeBardeleben, "TensorFI: A flexible fault injection framework for tensorflow applications," in 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2020, pp. 426–435.

[10] Z. Chen, N. Narayanan, B. Fang, G. Li, K. Pattabiraman, and N. DeBardeleben, "TensorFI: A flexible fault injection framework for TensorFlow applications," in Proc. Int. Symp. Softw. Rel. Eng., 2020, pp. 426–435.

[11] S. Hari, T. Tsai, M. Stephenson, S. Keckler, and J. Emer, "SASSIFI: An architecture-level fault injection tool for GPU application resilience evaluation," in Proc. Int. Symp. Perform. Anal. Syst. Softw., 2017, pp. 249–258.

[12] "SCE-mi (standard Co-Emulation Modeling Interface)" Accellera Systems Initiative. https://www.accellera.org/downloads/standards/sce-mi (accessed Sep. 12, 2024).

[13] S. El-Ashry, M. Khamis, H. Ibrahim, A. Shalaby, M. Abdelsalam and M. W. El-Kharashi, "On Error Injection for NoC Platforms: A UVM-Based Generic Verification Environment," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 5, pp. 1137-1150, May 2020, doi: 10.1109/TCAD.2019.2908921.

[14] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," arXiv:1607.02533 [cs, stat], Feb. 2017, Available: https://arxiv.org/abs/1607.02533.

[15] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," arXiv.org, 2014. https://arxiv.org/abs/1412.6572.

[16] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learningapplied to document recognition," Proceedings of the IEEE, vol. 86, 1998, pp.2278-2324.

[17] S. Feng, J. Wu, S. Zhou and R. Li, "The Implementation of LeNet-5 with NVDLA on RISC-V SoC," 2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 2019, pp. 39-42, doi: 10.1109/ICSESS47205.2019.9040769.

[18] "Questa Advanced Verification," Siemens Digital Industries Software. https://eda.sw.siemens.com/en-US/ic/questa/ (accessed Sep. 12, 2024).

[19] "Veloce CS system - IC emulation and prototyping" Siemens Digital Industries Software. https://eda.sw.siemens.com/en-US/ic/veloce/ (accessed Sep. 12, 2024).