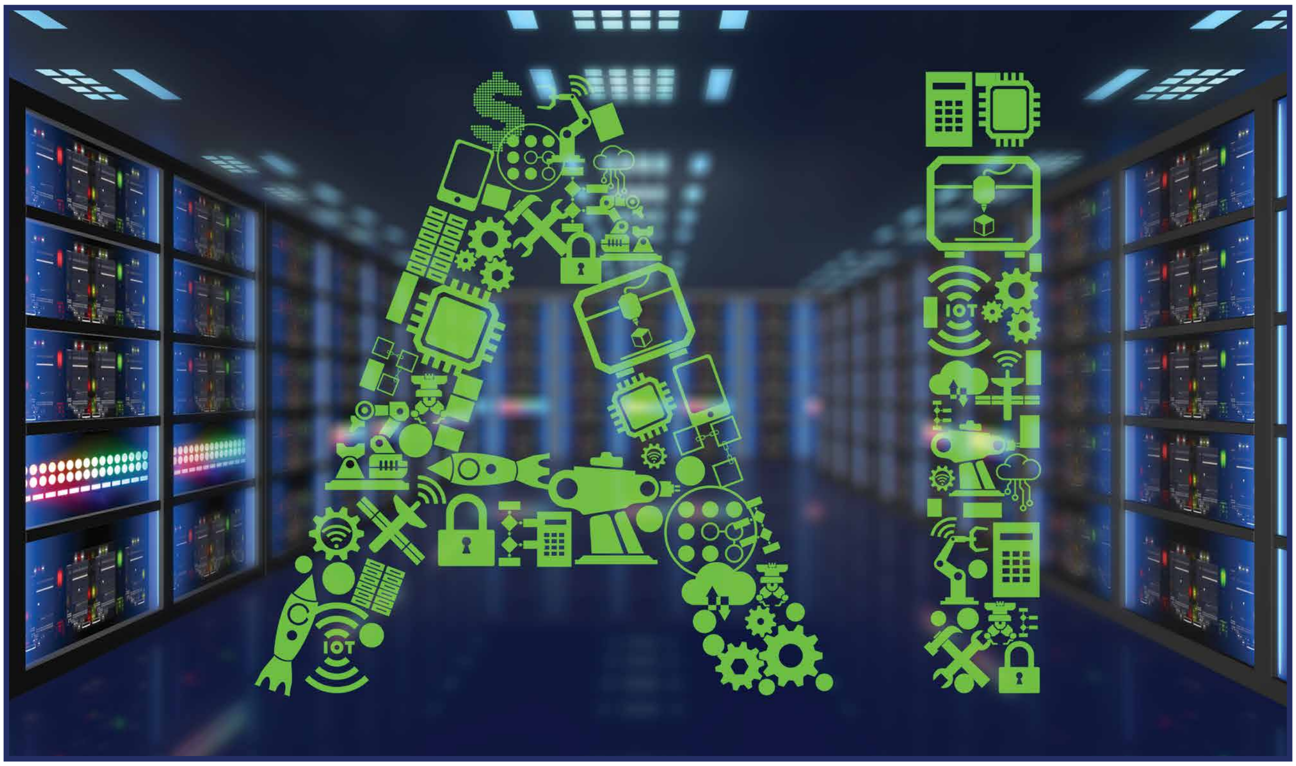
IEEE

MARCH/APRIL 2023

Design&Test



Special Issue on Testability and Dependability of Artificial Intelligence Hardware

CEDA IEEE Council on Electronic Design Automation

CAS IEEE CIRCUITS AND SYSTEMS SOCIETY

SSCS

tttc™

IEEE Design&Test

Submission Information: Submit a Word, pdf, text, or PostScript version of your submission to ScholarOne Manuscripts, https://mc.manuscriptcentral.com/dandt

Editorial: Unless otherwise stated, bylined articles and columns, as well as product and service descriptions, reflect the author's or firm's opinion. Inclusion in *IEEE Design&Test* does not necessarily constitute endorsement by IEEE. All submissions are subject to editing for style, clarity, and length.

IEEE Design&Test (ISSN 2168-2364) is published bimonthly by the Institute of Electrical and Electronic Engineering, Inc. Responsibility for the contents rests upon the authors and not upon the IEEE, the Society/Council, or its members. IEEE Corporate Office: 3 Park Avenue, 17th Floor, New York, NY 10016. IEEE Operations Center: 445 Hoes Lane, Piscataway, NJ 08854. NJ Telephone: + 1 732 981 0060. Price/Publication Information: To order individual copies for members and nonmembers, please email the IEEE Contact Center at contactcenter@ieee.org. (Note: Postage and handling charge not included.) Member and nonmember subscription prices available upon request. Copyright and Reprint Permissions: Abstracting is permitted with credit to the source. libraries are permitted to photocopy for private use of patrons, provided the per-copy fee of $31.00 is paid through the Copyright Oearance Center, 222 Rosewood Drive, Danvers, MA 01923. For all other copying, reprint, or republication permission, write to Copyrights and Permissions Department, IEEE Publications Administration, 445 Hoes Lane, Piscataway, NJ 08854. Copyright©2023 by The Institute of Electrical and Electronics Engineers, Inc. All rights reserved. Periodicals Postage Paid at New York, NY and at additional mailing offices. Postmaster: Send address changes to *IEEE Design&Test*, IEEE, 445 Hoes Lane, Piscataway, NJ 08854-4141. GST Registration No. 125634188. CPC Sales Agreement #40013087. Return undeliverable Canada addresses to: Pitney Bowes IMEX, P.O. Box 4332, Stanton Rd., Toronto, ON M5W 3J4, Canada. IEEE prohibits discrimination, harassment, and bullying. For more information visit http://www.ieee.org/nondiscrimination. Printed in U.S.A.

Contents

March/April 2023
Volume 40 Number 2

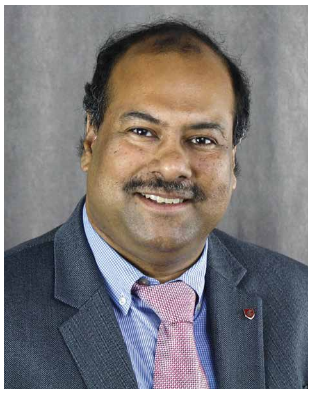
Copublished by the IEEE Council on Electronic Design Automation, the IEEE Circuits and Systems Society, the IEEE Solid-State Circuits Society, and the Test Technology Technical Council

ISSN: 2168-2364

Image credit: Image licensed by Ingram Publishing.

IEEE Design&Test

From the EIC



Special Issue on Testability and Dependability of Artificial Intelligence Hardware

The articles in this issue are divided into two groups: 1) the first group comprises articles that make up the Special Issue on Testability and Dependability of Artificial Intelligence Hardware and 2) the second group consists of general interest articles.

The highlight of this issue is the Special Issue on Testability and Dependability of Artificial Intelligence Hardware. There has been plethora of recent investigations on designing novel hardware architectures for artificial intelligence/machine learning (AI/ML) applications. Though achieving high performance and energy efficiency for the hardware architecture is of paramount importance, testability and dependability of these new architectures need to be addressed before the mainstream adoption. This special issue consists of seven articles from both academia and industry addressing the broad topic of testability and dependability of emerging AI hardware architectures. We thank the guest editors, Fei Su, Chunsheng Liu, and Haralampos-G. Stratigopoulos, for making this special issue possible.

In addition, we present three general interest articles, titled as follows: 1) "Using STLs for Effective In-Field Test of GPUs"; 2) "T-Topology Coupler-Based Bandpass Negative Group Delay Active Circuit Design and Test"; and 3) "FPGA-Chain: Enabling Holistic Protection of FPGA Supply Chain With Blockchain Technology."

This issue also contains a report on the 2022 International Conference on Computer-Aided Design (ICCAD) written by Tulika Mitra.

Many thanks to Scott Davidson for The Last Byte article titled "Is There an Answer?"

I hope you enjoy reading this issue of *IEEE Design&Test*.

■

[illegible]

Partha Pratim Pande, *Editor-in-Chief*
Washington State University
Pullman, WA 99164-2752 USA

Digital Object Identifier 10.1109/MDAT.2023.3243862
Date of current version: 10 March 2023.

Special Issue on Testability and Dependability of Artificial Intelligence Hardware

Fei Su
Intel Corporation
Folsom, CA 95630 USA

Chunsheng Liu
Alibaba Inc.
Sunnyvale, CA 94085 USA

Haralampos-G. Stratigopoulos
Sorbonne Université
French National Center for Scientific Research (CNRS), LIP6 Laboratory
75005 Paris, France

Artificial intelligence (AI) hardware, including AI accelerators and neuromorphic computing processors, emerges as one new frontier in the field of computing. There is an expedited paradigm shift in embracing bold and radical innovation of computer architectures, aiming at the continuation of computing performance improvement despite the slowed-down physical device scaling. Testability and dependability of AI hardware need to be addressed before mainstream adoption, especially in latency or throughput-critical, safety-critical, mission-critical, or remotely controlled applications (e.g., computer vision, autonomous driving, smart healthcare, IoTs, and robotics).

The guest editors of this Special Issue on Testability and Dependability of Artificial Intelligence Hardware have gathered manuscripts that cover innovative research from academia and industry for addressing the testability and dependability challenges of AI hardware arising from many aspects.

This special issue comprises one survey paper and six articles. The survey paper [A1] from the guest editorial team covers the state-of-the-art in research and development of dependability and testability solutions for AI hardware including digital or analog implementations of artificial neural networks (ANNs) and spiking neural networks (SNNs), used in accelerators and neuromorphic designs. Trends, challenges, and perspectives are also discussed in this article.

In [A2], Ozen and Orailoglu point out that the rules that govern the error resilience problem in neural networks deviate from those in general-purpose computing. While deep neural networks (DNNs) may inherently tolerate minor perturbations, there exists the potential vulnerability caused by a large-magnitude hardware error. This article presents a method where neural networks can learn and construct self-checking mechanisms to detect and suppress such large magnitude errors.

In [A3], Bertoa et al. present selective triple modular redundancy (TMR), an automated tool that analyzes the sensitivity of computations within neural network inference to the overall network accuracy. The tool then triplicates the most sensitive computations to increase the functional safety of the neural network accelerator, without resorting to full TMR.

Digital Object Identifier 10.1109/MDAT.2023.3241114
Date of current version: 10 March 2023.

The proposed method allows designers to explore the tradeoff between accelerator reliability and hardware cost.

In [A4], Omland et al. propose an application program interface (API)-based method for hardware fault simulation to investigate the effect of hardware fault on DNN output failure probability for common DNN accelerators. In a proof of concept presented in the article, speed ups of the order of >100 compared to full hardware simulations have been achieved.

Silicon-photonic neural networks (SPNNs) are being explored as post-Moore's law successors to CMOS-based AI accelerators, thanks to their ultrahigh speed and ultralow energy consumption. However, their accuracy and energy efficiency can be catastrophically degraded because of the sensitivity of underlying photonic components to fabrication process variations and run-time uncertainties. In [A5], Banerjee et al. present a method of criticality assessment to identify susceptible components of SPNNs. The results show that the criticality of uncertainties varies significantly based on both the location and the tuned characteristics of the affected components.

In [A6], Kundu et al. focus on the reliability of DRAM utilized as the main memory subsystem in sparse DNN accelerators. Their analysis shows a single fault in the encoded memory compression bitmap causes a significant accuracy reduction in classification applications. The authors present a systematic quality-aware mitigation strategy with a low memory overhead.

In the last article of this special issue [A7], Paul et al. study read disturb failures of a nonvolatile memory (NVM) cell used in neuromorphic hardware to store model parameters. The authors propose a system software framework to incorporate the insights from analysis in programming model parameters on NVM cells to mitigate the read disturbances.

We hope readers will enjoy reading these articles. We also hope any insights from this special issue will inspire more researchers from academia and industry to take a new adventure in this new field.

We would like to thank all the authors who contributed to this special issue, the reviewers for providing constructive feedback, the past Editor-in-Chief Prof. Jörg Henkel, the current Editor-in-Chief Prof. Partha Pratim Pande, the Associate Editor-in-Chief Prof. Mehdi Tahoori, as well as the editorial staff of *IEEE Design&Test* for making this special issue possible. ■

Appendix: Related Articles

[A1] F. Su, C. Liu, and H.-G. Stratigopoulos, "Testability and dependability of AI hardware: Survey, trends, challenges, and perspectives," *IEEE Des. Test*, vol. 40, no. 2, pp. 8–58, Mar. 2023.

[A2] E. Ozen and A. Orailoglu, "Shaping resilient AI hardware through DNN computational feature exploitation," *IEEE Des. Test*, vol. 40, no. 2, pp. 59–66, Mar. 2023.

[A3] T. G. Bertoa et al., "Fault-tolerant neural network accelerators with selective TMR," *IEEE Des. Test*, vol. 40, no. 2, pp. 67–74, Mar. 2023.

[A4] P. Omland et al., "API-based hardware fault simulation for DNN accelerators," *IEEE Des. Test*, vol. 40, no. 2, pp. 75–81, Mar. 2023.

[A5] S. Banerjee, M. Nikdast, and K. Chakrabarty, "On the impact of uncertainties in silicon-photonic neural networks," *IEEE Des. Test*, vol. 40, no. 2, pp. 82–89, Mar. 2023.

[A6] S. Kundu et al., "Analysis and mitigation of DRAM faults in sparse-DNN accelerators," *IEEE Des. Test*, vol. 40, no. 2, pp. 90–99, Mar. 2023.

[A7] A. Paul et al., "On the mitigation of read disturbances in neuromorphic inference hardware," *IEEE Des. Test*, vol. 40, no. 2, pp. 100–108, Mar. 2023.

Fei Su is a DFX and telemetry architect at Intel Corporation, Folsom, CA 95630 USA. His research interests include testability and dependability of semiconductor circuits/chiplets, artificial intelligence (AI)/ML hardware, cyber-physical systems, and edge/cloud computing. Su has a PhD from Duke University, Durham, NC, USA. He is a Senior Member of IEEE.

Chunsheng Liu is the leader of the DFT Team at Alibaba Inc., Sunnyvale, CA 94085 USA. His research interests include test infrastructure for high-performance processors, FPGA and machine learning accelerators, and high dependability of cloud computing hardware. Liu has a PhD from Duke University, Durham, NC, USA. He is a Senior Member of IEEE.

Haralampos-G. Stratigopoulos is a research director of the French National Center for Scientific Research (CNRS) at the LIP6 Laboratory, Sorbonne Université, 75005 Paris, France. His research interests include neuromorphic computing, hardware security, and design-for-test of integrated circuits and systems. Stratigopoulos has a PhD from Yale University, New Haven, CT, USA. He is a Member of IEEE.

■ Direct questions and comments about this article to Fei Su, Intel Corporation, Folsom, CA 95630 USA; fei.su@intel.com.

Testability and Dependability of AI Hardware: Survey, Trends, Challenges, and Perspectives

Fei Su
Intel Corporation
Folsom, CA 95630 USA

Chunsheng Liu
Alibaba Inc.
Sunnyvale, CA 94085 USA

Haralampos-G. Stratigopoulos
Sorbonne Université, CNRS,
LIP6 Laboratory
75005 Paris, France

Editor's notes:
Hardware realization of artificial intelligence (AI) requires new design styles and even underlying technologies than those used in traditional digital processors or logic circuits. Therefore, their dependability and testability requirements, challenges, and solutions are fundamentally different and unique. This survey article covers this very important topic and provides insight to such issues.
—Mehdi Tahoori, Karlsruhe Institute of Technology

■ **Artificial intelligence (AI)** and machine learning (ML) algorithms have been a subject of interest for several decades now. Although AI and ML have gone through hype cycles of disappointment and enthusiasm, recent algorithmic advancements, in particular, deep neural networks (DNNs) [1], as well as the availability of big data and the rapid growth of computing power, have renewed interest leading nowadays to applications in numerous distinct fields, for example, robotics, medicine, autonomous vehicles, computer vision, speech recognition, natural language processing, gaming, and so on.

DNN models are computational intensive, with their backpropagation training process taking up a number of operations in the order of millions. Inference on trained models requires a single forward pass, but still the number of operations remains very high. From a hardware perspective, this poses severe challenges to data storage, movement, and processing speed on conventional central processing units (CPUs) with a traditional Von Neumann computer architecture, commonly known as the memory wall problem [2]. To this end, there are intense and ongoing efforts nowadays toward designing dedicated and customized processors for AI [3], [4], [5], [6], [7], [8], [9], referred to as AI hardware accelerators, which belong to the larger family of domain-specific computing paradigms. Widely used AI hardware accelerators today are graphics processing units (GPUs) and FPGAs, but orders of magnitude of energy-speed improvement can be achieved with ASICs.

Digital Object Identifier 10.1109/MDAT.2023.3241116
Date of publication: 31 January 2023; date of current version: 10 March 2023.

Another high incentive for designing AI hardware accelerators is to push the execution of AI algorithms from the cloud closer to the sources of data onto edge devices [10]. This is driven by energy, bandwidth, speed, availability, and privacy requirements. More specifically, edge computing reduces the data transfer requirement, thus saving energy and bandwidth. Saving bandwidth is important given the forecast that several tens of billions of edge devices will be connected to the internet in the near future. Several applications, for example, autonomous vehicles, require low-latency real-time computation which is slowed down due to the communication with the cloud. Also, several applications require availability, and therefore they need to be less dependent on communication with the cloud. Finally, handling data locally offers privacy as opposed to transmitting sensitive data over the cloud. Edge AI is a challenging objective since edge devices have limited resources and are often battery-operated. Typically, AI hardware accelerators embedded on edge devices perform only inference with the DNN model trained in software and uploaded upfront.

Having stressed that AI hardware accelerators are pivotal in the AI world, many would believe that neural networks on hardware inherit the remarkable fault-tolerant capabilities of the biological brain. Indeed, biological neural networks are capable of regenerating, rewiring, or adapting network elements to make up for the damage, which is part of their neuroplasticity ability [11]. This assumption also stems from properties of neural networks, such as their high parallelism and over-provisioning, that is, there are more neurons available than the minimum required for a certain cognitive task and many neurons end up being ineffective. However, as it will be discussed in more detail in a later section, recent hardware-level fault injection experiments have shown that this assumption is false. A neural network is likely to be capable of learning even in the presence of a high fault rate; however, the impact on prediction accuracy can be nonnegligible or even detrimental if a model is uploaded on a faulty hardware neural network or if a fault occurs during the lifetime of the hardware neural network.

For these reasons, the testability and dependability of AI hardware accelerators are important issues that need to be addressed already from the design phase [12]. Inspiration can, of course, be drawn from known and mature methodologies applied to traditional computer architectures, but the architectural particularities of AI hardware accelerators often make such methodologies prohibitive in terms of cost and quality, requiring the development of new methodologies that are better suited and take full advantage of the said architectural particularities.

The aim of this article is to provide a survey of existing works on testability and dependability methodologies for AI hardware accelerators and discuss trends, challenges, and perspectives. The high-level organization of this article into sections and their main subsections is shown in Figure 1.

Neural network types

We distinguish two types of neural networks, namely artificial neural networks (ANNs) and spiking neural networks (SNNs). Both are inspired by the brain structure composed of layers assembled by neurons and synapses interconnecting the different layers. The term "deep" in DNNs refers to the number of layers going beyond just a few, allowing to extract

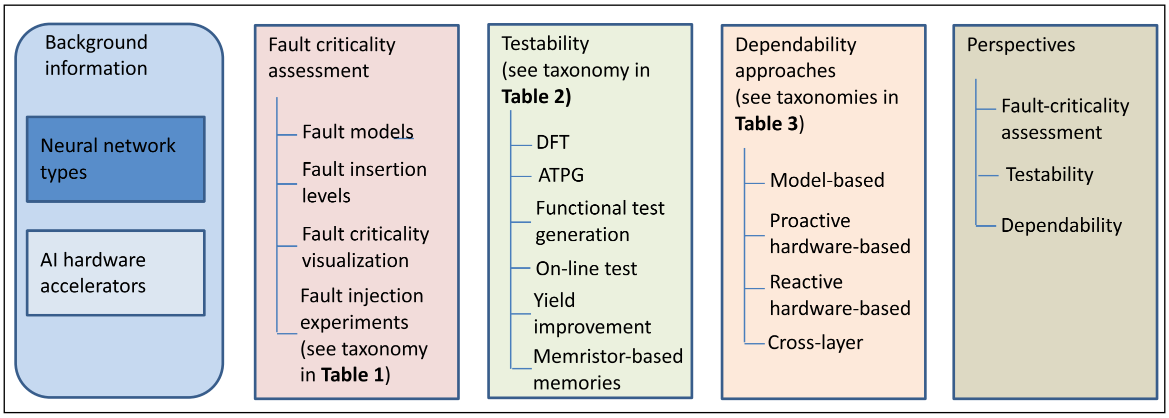


Figure 1. Structure of this article.

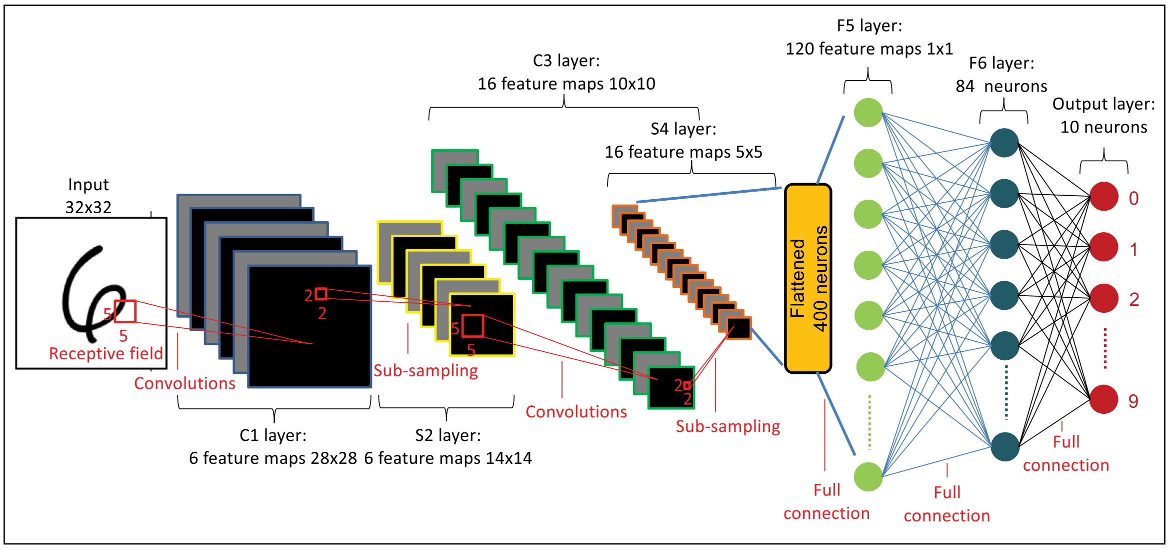


Figure 2. Architecture of LeNet-5 CNN.

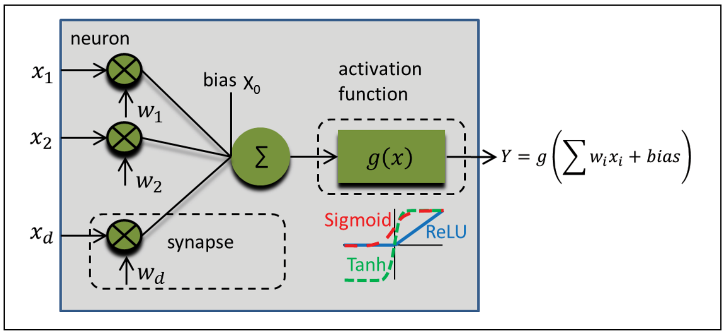


Figure 3. Artificial neuron.

more complex features. The number of layers, the number of neurons within each layer, and synapse connections define a network topology.

There are three main topologies applied to both ANNs and SNNs: 1) fully connected (FC) networks; 2) convolutional neural networks (CNNs) [13]; and 3) recurrent neural networks (RNNs) [14]. Figure 2 shows an example CNN with FC layers forming the last layers. In FC networks, the neurons of a new layer are connected via synapses to the outputs of all neurons in the prior layer. In CNNs, a convolutional layer is composed of several feature maps. A feature map is a plane of neurons where each neuron is connected to the outputs of spatially nearby neurons contained in a lower dimensional plane of the prior layer, referred to as a receptive field. Each neuron has a different receptive field located at different coordinates of the prior layer. In a given feature map, all neurons are constrained to share the same synaptic weights, whereas synaptic weights change from one feature map to another. Convolutional layers are alternated with subsampling layers which are used to downsample the output of the preceding convolutional layer. There are different types of subsampling, such as max pooling and average pooling. Max pooling captures the maximum value of the receptive field and processes it to the output, whereas average pooling calculates the average value. CNNs allow synapse reuse and reduce the number of synapses compared with an FC network. In RNNs, neurons can additionally receive as input their previous state or the previous state of a neuron in a subsequent layer, thus realizing an internal memory retaining past information to forecast future outputs. RNNs are used for learning on time-series or sequential data, while FC networks and CNNs are feed-forward and inputs are independent of each other.

In ANNs, data are represented as static numerical values. Neurons apply a nonlinear activation function, such as rectified linear unit (ReLU), sigmoid, and tanh, on the weighted sum of outputs of other neurons, as depicted in Figure 3. The weights are scalar values and correspond to the synaptic weights.

In SNNs, on the other hand, data are represented with spikes processed in a continuous way in time, which is similar to brain operation. Thus, they are more biologically plausible compared with ANNs, bridging the gap between the ML and the biological brain in terms of computation speed and power consumption [15]. SNNs form the basis of neuromorphic computing as pioneered by Mead [16]. The most hardware-friendly spiking neuron implementation is the integrate & fire (I&F) model [17], depicted in Figure 4. The neuron integrates the spikes from incoming synapses, and when the potential of its membrane exceeds a threshold, it fires a spike of its own that propagates through synapses to other neurons. It also resets the threshold so as to be able to fire again. The neuron has two additional brain-inspired functionalities. It has a refractory period, that is, it is allowed to fire only if a certain time is elapsed since the last output spike, and a leakage behavior, that is, the membrane potential decreases between two consecutive input spikes. The synapse operation is different from ANNs and also resembles the biological synapse operation. A synapse receives spikes and in turn stimulates the membrane potential of postsynaptic neurons via a current. The most common information representation in SNNs is rate coding, whereby the information is encoded into the firing rate over an observation period, but other representations have been suggested, including time-to-first-spike and interspike interval.

From a hardware perspective, there is a belief that SNNs offer faster inference and lower energy consumption compared with ANNs. This belief stems from two SNN characteristics, namely the real-time asynchronous spike flow and the sparsity of the spike flow, which reduces neuron activities. In contrast, ANNs have a frame-based operation, that is, for a layer to perform its computation, the layer has to wait for the computation of the previous layer to complete and every individual neuron is being evaluated. However, SNNs are harder to train compared with ANNs due to the noncontinuity of the spiking neuron's transfer function, as well as the additional parameters a spiking neuron carries, for example, threshold, leakage rate, refractory period, which could be sensitive. In general, the discussion on the relative performance between ANNs and SNNs is not trivial due to the different input type, that is, the sequence of static frames versus continuous-time event flow. Converting a data set from frame-based to spiking format and vice versa creates a bias in the comparison. In general, the advantage of one neural network type over the other is task-dependent, with the SNNs being ideally suited for processing spatiotemporal event-based sensory data. For an extensive discussion on SNNs and the comparison with their ANN counterparts, the readers are referred to [5], [18], and [19].

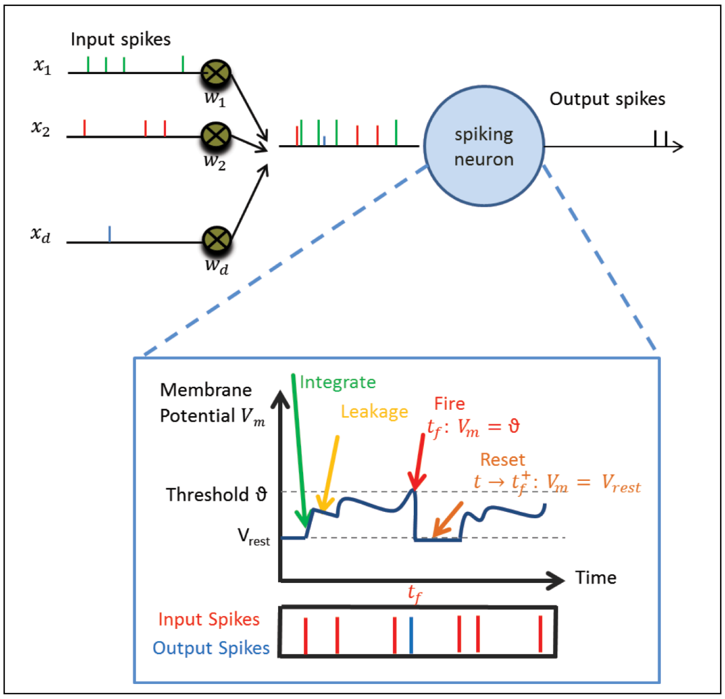


Figure 4. Spiking neuron.

AI hardware accelerators

Silicon implementations of neural networks appeared decades ago with early efforts demonstrating few-layers, few-neurons per layer networks [20]. Moving to larger designs for DNN acceleration, the main challenge is the memory wall that limits the throughput and increases power consumption. The design ambition is, therefore, to overcome the memory wall by distributing the memory within close proximity to the processing elements (PEs), for example, the multiple–accumulate (MAC) units, or through the interleaving of memory and PEs. Basic architectures include the streaming architecture composed of many cores with the layers mapped among the cores and the single-core architecture, that is, in the form of a systolic array that parallelizes the storage and computation of the different layers [3]. SNNs typically employ the streaming architecture with a core receiving and transmitting spikes via the address event representation (AER) protocol that essentially implements a network-on-chip (NoC) communication scheme [21]. Clearly, efficient mapping of the neural network algorithm onto the hardware becomes of utmost importance and different neural network topologies require different hardware designs to fully take advantage of neuromorphic computing.

Analog and mixed-signal (AMS) implementations can offer orders of magnitude lower power consumption compared with their digital counterparts, and therefore they are better-suited for edge computing being capable of acting directly on sensory data from the world–machine interfaces [22], [23]. This is because transistors are operated in the subthreshold region, and the main operations of a neural network, that is, addition and multiplication, can be performed efficiently in the analog domain. Addition can be performed using Kirchhoff's current law, while multiplication can be performed with just a few transistors. However, they are less robust due to process variations and noise.

One way to reduce energy consumption is approximate computing that involves two strategies. The first uses approximate arithmetic units in the PEs [24]. The second is termed network compression or quantization [25]. It reduces the precision of the weights and neuron activation values by transforming floating-point numbers into narrow few-bit integers. At the extreme, this results in binary neural networks (BNNs) that use 1-bit precision [26], further simplifying the network architecture by using XNORs instead of MAC units [27]. BNNs save energy and storage and can serve for implementing deep models in resource-constrained edge devices. Network compression results in accuracy loss but it may be recovered through training.

Another design paradigm with tremendous potential for overcoming the memory wall is in-memory computing where the matrix–vector multiplications are performed within the memory itself [28], [29]. In-memory computing has two main embodiments, namely performing arithmetic and logic operations within the SRAM or using memristive crossbar arrays.

A memristive crossbar array is composed of horizontal and vertical metal lines with a memristive device placed at each cross-point intersection connecting the two metal lines, as shown in Figure 5. The conductance of the memristive device implements the synapse weight, horizontal lines are driven by the voltage output of presynaptic neurons, and vertical lines provide the current input of postsynaptic neurons. Each column implements the dot product $I_i = \Sigma_j G_{i,j} \cdot V_j$ and parallelized dot-products across the columns implement efficient in-situ matrix-vector multiplication $I = \mathbf{G} \cdot V$ in analog form, reducing computational complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(1)$. Each memristive device is augmented with an access device, as shown in the 1-transistor/1-resistor (1T1R) architecture of

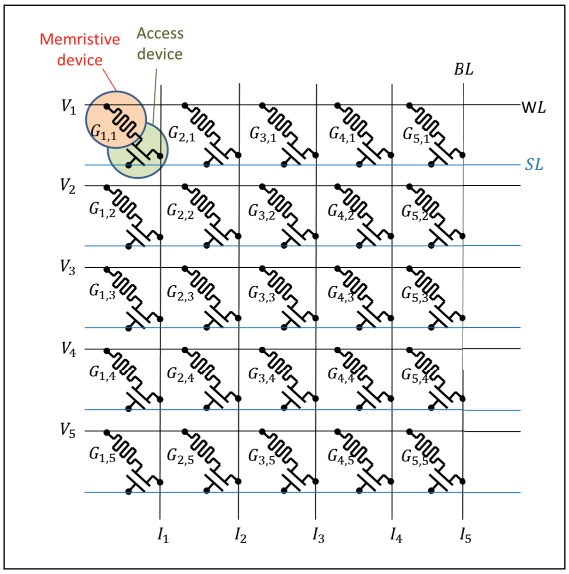

Memristive device
Access device
BL
WL
SL
V_1
V_2
V_3
V_4
V_5
$G_{1,1}$ $G_{2,1}$ $G_{3,1}$ $G_{4,1}$ $G_{5,1}$
$G_{1,2}$ $G_{2,2}$ $G_{3,2}$ $G_{4,2}$ $G_{5,2}$
$G_{1,3}$ $G_{2,3}$ $G_{3,3}$ $G_{4,3}$ $G_{5,3}$
$G_{1,4}$ $G_{2,4}$ $G_{3,4}$ $G_{4,4}$ $G_{5,4}$
$G_{1,5}$ $G_{2,5}$ $G_{3,5}$ $G_{4,5}$ $G_{5,5}$
I_1 I_2 I_3 I_4 I_5

Figure 5. Memristive crossbar array.

Figure 5, that allows selecting a memristive device for programming while not disturbing the stored state of other memristive devices. A memristive crossbar array is accompanied by peripheral circuits (not shown in Figure 5) if communication between crossbars is implemented in the digital domain. These include digital-to-analog converters (DACs) and analog-to-digital converters (ADCs), which contribute a large fraction of the area and power consumption of the array macro.

There are several emerging nonvolatile memory (NVM) devices that can be used to implement the memristive device, including resistive random access memory (ReRAM), phase change memory (PCM), and spin transfer torque magnetic random access memory (STT-MRAM) [30]. These devices are compact and can perform read and write operations with low power. However, they suffer from several imperfections, such as conductance variations and drifts, which result in poor yield, stability, and endurance. Therefore, enhancing the reliability of crossbar-array computation is a subject of ongoing research.

Finally, 3-D integration technologies could offer several advantages such as short interconnections, high parallelism, high bandwidth, and small form factors [31], [32].

A taxonomy of AI hardware accelerators is illustrated in Figure 6. The first layer defines the type of neural network, that is, ANN or SNN. The second layer defines different design flavors, that is, digital or AMS, with in-memory computing inserted as a separate category. The third layer defines the

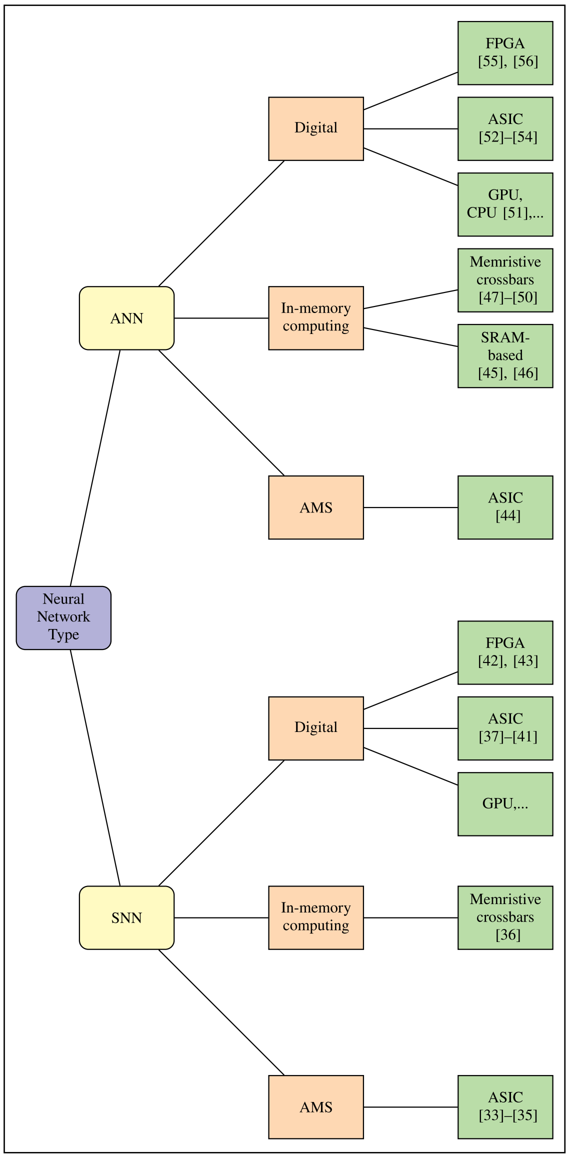


Figure 6. Taxonomy of AI hardware accelerators.

implementation, that is, ASIC, FPGA, and so on, while in-memory computing is further distinguished into digital processing-in-memory, that is, SRAM-based in-memory computing, and memristive crossbar arrays. In Figure 6, we provide references to representative designs with a focus on designs that have been demonstrated on silicon. Regarding memristive crossbar-array accelerators, most works present only simulation results up to now. This list of references is not meant to be complete. For recent and thorough surveys on accelerator design for ANNs and SNNs, the readers are referred to [3], [4], [6], [7], [8], [9], and [5], respectively.

Fault criticality assessment

Introduction

In the context of an AI hardware accelerator, many faults turn out to be benign: they are masked before their effect reaches the output or produce an output change that is tolerable, that is, it does not translate to performance loss. This is thanks to the network sparsity, the over-provisioning, the distributed computing, and the nature and sequence of mathematical computations. Some faults, however, will be critical and will affect the performance. A fault classification is shown in Figure 7. The ability to quickly assessing the impact of faults on the AI hardware accelerator performance is very valuable for performing early reliability analysis and for guiding the development of efficient and cost-effective fault detection, fault tolerance, and fault repair schemes by placing the focus on targeting the critical faults only.

A network is viewed as a distributed system where neurons and synapses can fail independently [58]. Given a large number of synapses and neurons, the size of the fault space easily explodes, not to mention the rest of the hardware components. On the other hand, the fault impact is typically expressed in terms of accuracy drop on the testing set, which can contain several thousands of samples, while the time for a single inference can be very long. For these reasons, performing fault simulation at the hardware level can be intractable, thus necessitating fault modeling approaches at a higher abstract level. As a matter of fact, performing exhaustive fault injection even on a higher abstract network representation may still not be feasible, thus necessitating fault sampling.

Another challenge in fault modeling is that the fault impact is determined by the interactions between the network model, the data set, and the AI hardware accelerator. When analyzing the fault impact, the AI hardware accelerator architecture and the scheduling of network operations on its architectural components cannot be ignored [59].

Figure 8 shows a fault injection experiment flow. Starting with a fault model, a fault list is created as a subset of the fault universe possibly using fault sampling. A single-fault assumption or multiple-fault scenario with a user-specified fault rate can be considered in this step. Then fault injection is performed on the AI hardware accelerator which could be

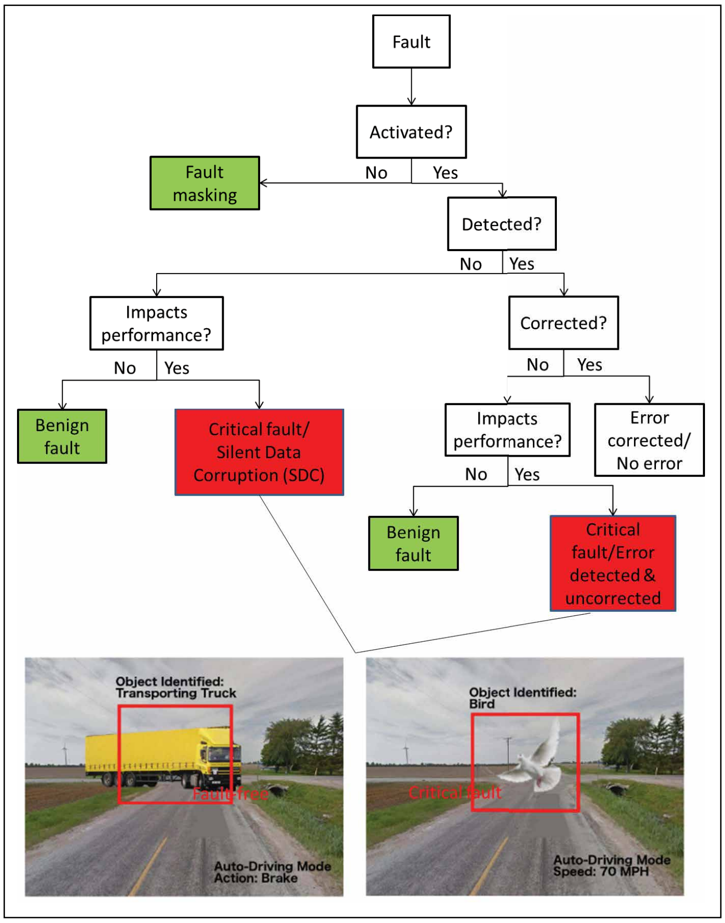

Fault
Activated?
No
Yes
Fault masking
Detected?
No
Yes
Impacts performance?
Corrected?
No
Yes
No
Yes
Benign fault
Critical fault/ Silent Data Corruption (SDC)
Impacts performance?
Error corrected/ No error
No
Yes
Benign fault
Critical fault/Error detected & uncorrected
Object Identified: Transporting Truck
Fault-free
Auto-Driving Mode Action: Brake
Object Identified: Bird
Critical fault
Auto-Driving Mode Speed: 70 MPH

Figure 7. Fault classification. Images are from [57].

done at different insertion levels, that is, in a software model, RTL level, microarchitectural level, gate level, transistor level, on an actual hardware prototype, or with radiation. For every fault scenario, the fault impact is assessed and stored. After going through the complete fault list, a report is produced, for example, including the benign and critical faults, the critical fault locations, and the fault rate that can be tolerated.

Examples of fault criticality visualizations are shown in Figure 9. In Figure 9a, the *x*-axis shows the different layers and for each layer, there are two columns, each corresponding to a different fault type. A column is a colored bar possibly separated into chunks of different colors. Each chunk of the bar corresponds to a specific classification accuracy according to the color shading shown at the bottom of Figure 9, and the projection on the *y*-axis shows the percentage of neurons for which the fault results in this classification accuracy. While Figure 9a shows the cumulative neuron criticality across layers, Figure 9b shows the per-neuron criticality as a heat map with the neuron number in the *x*-axis and the layer number in the *y*-axis. Each orthogonal corresponds to one specific neuron, and the color of each orthogonal corresponds to the classification accuracy in the presence of a fault in the neuron according to the color shading at the bottom of Figure 9. Figure 9c displays the impact on classification accuracy of synapse faults in the synaptic matrix between two layers. Using such plots, one can label faults as critical or benign and identify critical fault locations across layers and within each layer.

As we will see next, most research works consider bit-flips in the memories and registers storing the network parameters, that is, synapse weights and neuron activations. For this fault model, examples of reliability assessment are shown in Figure 10. Bit-flips can be injected with some bit error rate (BER) probability to assess the largest BER that can be tolerated, as shown in Figure 10a. The experiment is repeated several times, and summary statistics are visualized in Figure 10a using box plots. The bottom and top edges of the box indicate the 25th and 75th percentile, respectively. The whiskers extend to the most extreme data points without considering outliers, and the outliers are plotted individually using the "o" symbol and are not always aligned vertically for illustration purposes. Figure 10a also illustrates the baseline fault-free accuracy shown with the green zone, the median shown with a dotted circle, and the average accuracy across repetitions of the same experiment shown with a red line. Bit-flips can also be injected at individual bit positions as shown in Figure 10b where the network parameter has an 8-bit representation. For example, with the results in Figure 10b, we can identify those bits starting from the least significant bit (LSB) that have no impact on the accuracy if they are flipped and can be left unprotected in a fault-tolerant strategy.

In the upcoming section, we survey several works demonstrating fault injection experiments and frameworks. Figure 11 shows different fault types at different insertion levels. A taxonomy of works is provided in Table 1 based on the fault insertion level, while memristor crossbar-based architectures are treated as a special category. In the upcoming section, we summarize the general conclusions from these experiments.

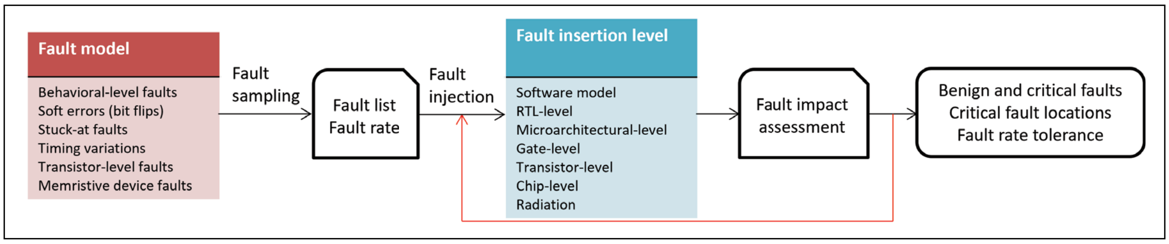


Figure 8. Fault injection experiment flow.

Fault injection experiments

Software level

The software and hardware implementation of a neural network matches closely in terms of component connectivity and data flow, thus allowing performing fault injection in software in a more time-efficient manner. This was noticed in early works [58], [61], [62], [63] where structural behavioral-level fault models were used in the main software operators that support the network computational task, that is, neurons and synapses. Behavioral-level fault types included stuck-at nodes, missing or saturated neurons, errors in the summation or the evaluation of the neuron's nonlinear activation function, errors in synaptic multiplication, disabled or saturated weights, errors in learning rules, noisy inputs, and so on. These behavioral-level faults can be mapped to physical fault models and root causes in hardware, that is, gate-level stuck-at faults and soft errors, for both digital and analog circuit implementations of neural networks [64], [65]. In [58], a theoretical study is presented for feed-forward neural networks (FFNNs) deducing the number of failing neurons and synapses an FFNN can tolerate.

As there is a large body of work in this direction for modern AI hardware accelerators, we categorize them according to the two neural network types, namely ANNs and SNNs.

ANNs

In [57], the fault model used is bit-flips in datapaths and buffers. A wide range of data types are considered, and bit flips are injected in different bit positions. Fault injection is carried out in the open-source DNN simulator framework Tiny-CNN written in C++, where each line of the code is mapped to the corresponding hardware component so as to

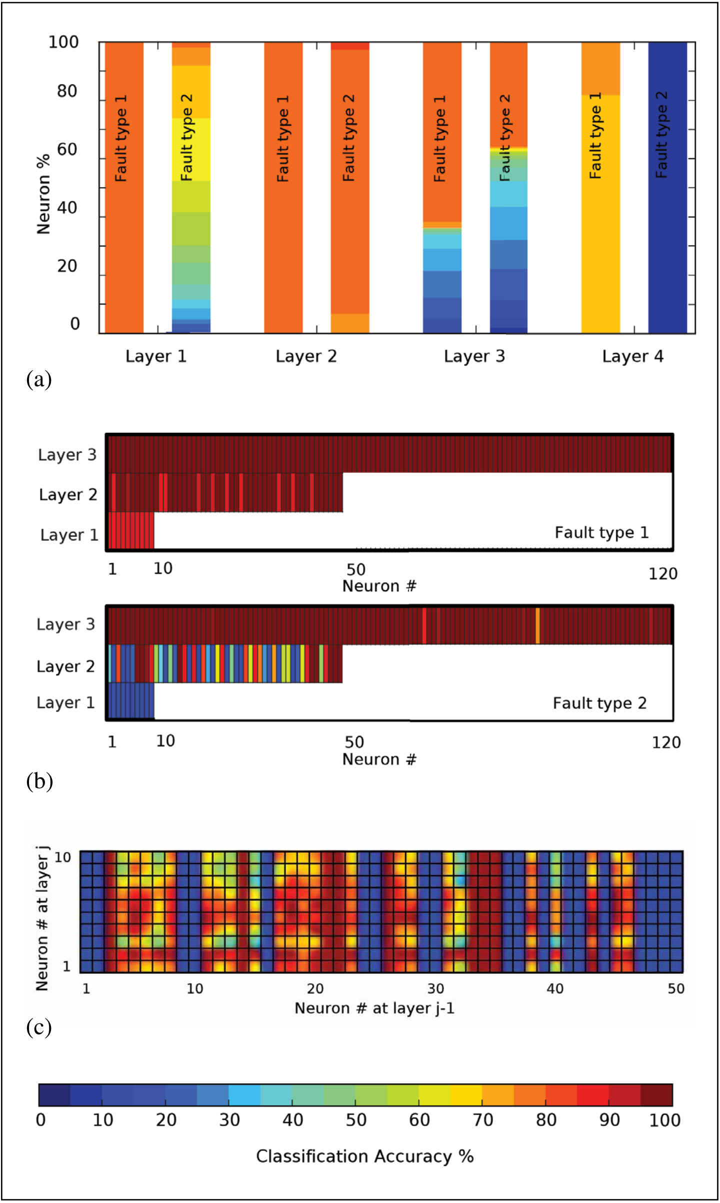


Figure 9. Fault criticality visualization. (a) Cumulative neuron fault criticality across layers. (b) Per-neuron fault criticality. (c) Synapse fault criticality.

pinpoint the impact of the fault injection location in terms of the underlying microarchitectural components. The focus is on CNNs considering different image classification tasks. Different types of silent data corruption (SDC), defined as a mismatch between the output of a faulty and the fault-free inference execution, or "fault ratings" are proposed taking into consideration that networks may rank predictions based on a confidence score. Some conclusions of this large-scale fault injection study are: 1) different DNNs have different sensitivities to SDCs depending on the topology, the types of layers, the data type used, and the position of the bit flip; 2) failure in-time (FIT) rates can exceed the safety standards, for example, ISO 26262 for automotive, by orders of magnitude; 3) data types that provide more dynamic value range are more vulnerable to SDCs since there are likely to be redundant value ranges that lead to larger-value deviation under faults. This implies that just-enough numeric value range and precision is advantageous from a reliability point of view; and 4) normalization layers reduce the impact of faults by averaging fault values with adjacent correct values.

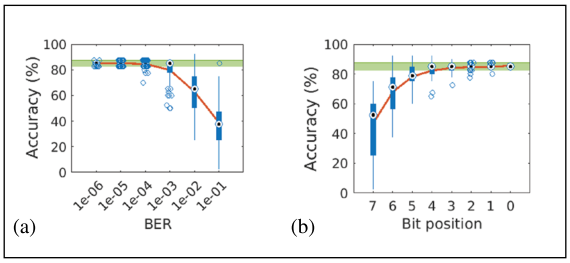


Figure 10. Reliability assessment using bit-flips as a fault model. (a) Accuracy drop for bit-flips with different BER levels. (b) Accuracy drop for bit-flips at different positions of the word representing a network parameter.

In [66], the *Ares* framework is proposed that simulates static bit-flips in the memory of the DNN accelerator. *Ares* is built on top of Keras [119],

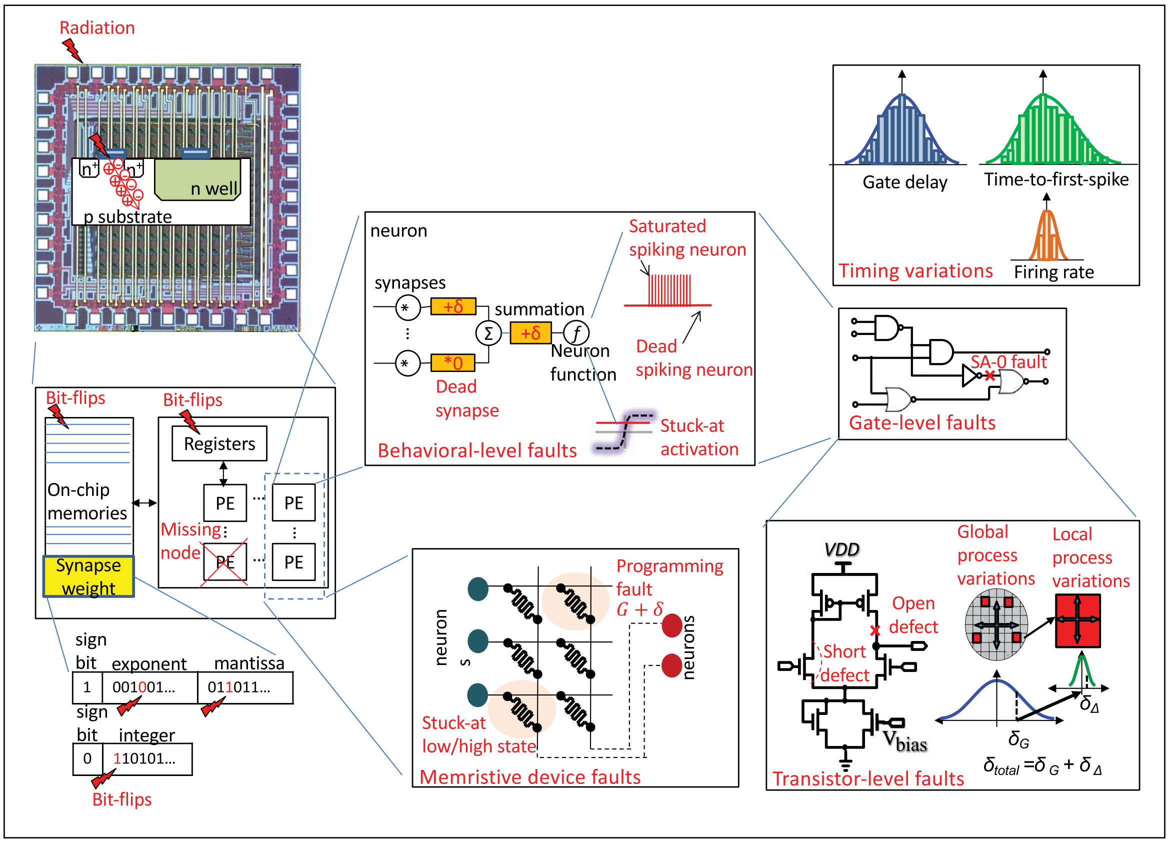


Figure 11. Faults models at different insertion levels. The chip image corresponds to an AMS implementation of an FC network used as an on-chip classifier for BIST purposes [60].

Table 1. Taxonomy of fault injection experiments and frameworks.

Fault injection experiments and frameworks
Software-level [57], [58], [61]–[63], [66]–[91]
RTL-level [92], [93]
Microarchitectural-level [94]
Gate-level [95]–[98]
Transistor-level [99], [100]
Chip-level [101]–[110]
Radiation experiments [81]–[85], [111]–[114]
Memristor crossbar-based architectures [115]–[118]

which takes high-level DNN descriptions specified in Python and executes them using either Theano [120] or TensorFlow [121] backends. Fault injection experiments are performed for several DNN models and data sets to study the classification rate as a function of BER. Fault injection is performed across the whole network, per-layer, and across network components, that is, weights and activation functions. The main conclusions of this study are: 1) a thresholded behavior is observed where for small BERs, the classification error is zero, but there is a BER threshold beyond which the classification error rises exponentially from zero; 2) there is a largely spread fault sensitivity or resilience across the DNN models, for example, the threshold varies by two orders of magnitude; 3) the weight quantization impacts resilience, that is, the larger the range of the possible weight values is the lower the threshold is; and 4) fault sensitivity across network layers and components can vary by several orders of magnitude.

In [67], the *FIdelity* DNN resilience analysis framework is proposed where hardware faults are modeled in software, that is, TensorFlow [121]; thereafter, high-speed software fault injection is performed. In this way, an analysis speedup is achieved while maintaining the level of accuracy of RTL or mixed-mode fault injection techniques. To map hardware faults in the software, the key insight is that hardware and software operations closely match, and all operations affected by a fault can be systematically derived, thanks to well-defined dataflow and scheduling algorithms. Given high-level architecture/hardware information and flip-flop (FF) FIT rate, the framework captures the effect of hardware faults to set a faulty output neuron using a reuse factor analysis for FFs. Faulty output neuron values are derived considering that each FF value already corresponds to a software-variable state. A key aspect of the framework is that it can treat logic transient errors in the data path and control FFs and not only memory errors.

In [68], a methodology is proposed to reduce the fault injection space and, thereby, the overhead of exhaustive fault injection. The underlying observation is that most ML functions in a DNN model, that is, convolution, ReLu, pooling, normalization, and so on, are monotonic. This means that in a word representing a model parameter, there exists an SDC-boundary bit such that bit-flips at higher-order bits would lead to SDCs and bit-flips at lower-order bits would be masked. Based on this observation, the binary fault injection (*BinFI*) fault simulator is proposed that bisects the fault injection space and finds the SDC-boundary bit with a binary-search-like algorithm. *BinFI* is built on top of the TensorFlow framework [121] duplicating the graph with customized operators.

In [69], a fault injection framework is proposed that reproduces fault models and event rates extracted from radiation tests. The ultimate goal is to have the flexibility of a software-based fault injector with a reliability assessment precision close to this of an accelerated neutron beam radiation-based fault injection experiment in a realistic harsh environment.

The interested reader is referred to [70], [71], [72], [73], [74], [75], [76], [77], [78], [79], [80], [81], [82], [83], [84], and [85] for more software-based fault injection experiments studying the fault impact for different neural network models, data type representations, layer types, network sizes, pruned networks, compressed versus uncompressed networks, and so on. Such experiments are also part of several other works that will be discussed in the upcoming sections and they are used for motivation or for guiding test and dependability solutions. Many of these works developed at the same time an in-house automated fault injection framework. The development of automated, fast, flexible, and accurate fault injection frameworks is an active area of research. Examples include PyTorchFI [86] and TensorFI [87], which are open-source and publicly available, and CLASSES [88]. An interesting research line is the development of ML-based frameworks that based on a small number of fault injections estimate the vulnerability for all parameters in the DNN in a short time [89]. Such approaches will be described in more detail in upcoming sections for systolic-array DNN architectures at the gate level and for memristor crossbar-array architectures. Another possibility is to use generic fault injection tools, such as SASSIFI [122], NVBitFI [123], and CAROL-FI [124], to emulate fault effects in the hardware platform, that is, GPU, running the application.

SNNs

The fault-tolerant characteristics of SNNs trained with different algorithms are studied in [90]. The fault model is a synapse fault where a faulty synapse is zeroed out or equivalently removed. Synapses are selected to be faulty at random with different failure rates. Results show that these different algorithms have different resilience characteristics. Resilience can greatly depend on the training algorithm and data set, and it can also show large variances according to the synapses that are selected to fail. A common conclusion is that for all considered networks, resilience drops rapidly as fault rates increase, and SNNs are not inherently resilient as it is frequently cited.

In [91], the behavioral-level fault model proposed in [100] (see the upcoming section) is used to perform accelerated fault injection in deep SNNs. The fault injection framework is built on top of the SLAYER [125] and PyTorch [126] frameworks by customizing the flow of computations and the faulty SNN is mapped onto a GPU. The general conclusion of this experiment is that saturation neuron faults are the most lethal and can severely affect inference regardless of the location of the neuron in the network and that the impact of all other fault types, that is, dead neuron faults and timing variations, may be severe only for neurons in the last hidden and output layer. At the extreme, timing variations could result in a dead or saturated neuron. In other words, a neuron that becomes permanently active has a greater effect on inference compared to a neuron that is permanently silenced or presents timing variations in its output spike train.

RTL level

In [92], fault characterization is performed on an RTL design of a typical accelerator. Fault injection is performed into the different registers that latch data during the inference, that is, input, weights, and intermediate layer computations. The fault model includes permanent stuck-at faults and transient faults occurring in a single random cycle. In each fault injection experiment, a different fault is randomly generated and injected by selecting a random register and set of bits and a random cycle in the case of transient faults. Fault characterization is performed across the different register types, layers, and components of fixed-point data representation, that is, sign, digit, and fraction, number of PEs, and network models. General conclusions are: 1) permanent faults are more critical than transient faults; 2) stuck-at-1 faults are more critical than stuck-at-0 faults due to the sparsity of zeros; 3) registers storing intermediate data are the most vulnerable, whereas input registers are the least vulnerable; 4) permanent faults are more critical in inner layers, that is, closer to the output, while the opposite is observed for transient faults; 5) sign, digit, and fraction are in this order more vulnerable; and 6) for permanent faults, the error decreases with the number of PEs, while there is no correlation in the case of transient faults.

In [93], an RTL-level fault injection framework is proposed that drastically reduces the fault simulation time. It makes use of a multilevel structure where on the lower level, the inference is split into several blocks corresponding to the neural network layers that run as standalone application processes, and on the upper level, these processes are synchronized.

Microarchitectural level

In software fault injection, the fault model risks being unrealistic and faults can be mapped to only a subset of hardware resources. In [94], the concept of two-level fault injection is adopted to evaluate the effects on CNN execution of faults in the GPU's scheduler and pipeline registers, two microarchitectural components that otherwise would be hidden in an abstract high-level CNN model. The microarchitectural simulation requires a prohibitively high time. To improve efficiency, the two-level fault injection idea is composed of the following steps: 1) perform microarchitectural fault injection, that is, transient bit-flips; 2) observe the effect on selected CNN tiles (i.e., matrix portions); 3) merge the corrupted tiles' output with the other tiles in the convolution considering their fault-free output to compose the layer's output; and 4) continue the execution of CNNs at the software level to check if the fault is eventually masked or it propagates at the output creating an error. Finally, a feedback analysis can determine the microarchitectural locations causing the observed critical errors that should be targeted for hardening.

Gate level

In [95], fault injection experiments are performed on a systolic array-based DNN accelerator. The core of the systolic array is composed of a 256 × 256 grid of MAC units. Each weight maps to exactly one MAC unit, thus a faulty MAC unit can result in multiple faulty weights. The systolic array is developed in Verilog and synthesized at the gate level. The fault model includes stuck-at faults at the gate level and timing faults created by under-scaling the power supply which essentially emulates process variations. It is demonstrated that training on a faulty systolic array can result in a significant classification drop when as few as four MAC units are faulty.

In [96], the impact of timing variations on the hardware implementation of ANNs is studied. Timing variations could result from delay defects, process variations, power supply noise, crosstalk, aging, voltage over-scaling, or frequency overclocking. Timing variations are modeled at the gate level by introducing an extra delay variation in the range of 10%–40% into each gate relative to the nominal gate delay. Results show that ANNs are sensitive to timing variations with the error growing larger as the timing variations worsen. Accuracy loss can be alleviated to a large degree but not fully recovered if the ANN is retrained under timing errors.

In [97] and [98], ML-based frameworks are proposed for analyzing the functional criticality of gate-level stuck-at faults in systolic array-based AI accelerators. Fault injection targets not only the interface/boundary level of a PE, but also all internal nodes of a PE. The main challenge of such a task is on computation/simulation overhead introduced by a significantly large number of potential fault injection points, for example, there will be tens of thousands of stuck-at faults for a single 32-bit PE, and billions for a 256 × 256 PE array. To this end, computationally efficient ML-based methods are proposed to speed up the analysis. The basic idea is based on the use of deep learning to predict fault criticality by utilizing the structural and data flow features. For example, in [98], a two-tier DNN-based model is presented, as illustrated in Figure 12. The first-tier DNN is trained on a data set obtained from ground-truth collection. The second-tier DNN is trained on a smaller and targeted data set containing the critical faults mispredicted as benign by the first-tier DNN. A generative adversarial network (GAN)-based method is further used to augment the data for the second-tier DNN, to minimize misclassification (i.e., misclassify critical faults as benign). The transferability of the proposed method is also investigated (i.e., if a fault criticality model trained on a PE can be transferred to evaluate a different PE). The results show that there exists some inherent transferability across PEs in the same array, mainly due to their identical topologies. On the other hand, more model retraining will be needed if transferability is not met.

Transistor level

Transistor-level fault simulations can be performed only at the neuron level or for small-sized networks.

In [99], transistor-level short- and open-circuit defects are injected into the fundamental logic operators of a perceptron, that is, adders and multipliers. Fault injection experiments in a shallow two-layer classical fully digital spatial expansion ANN

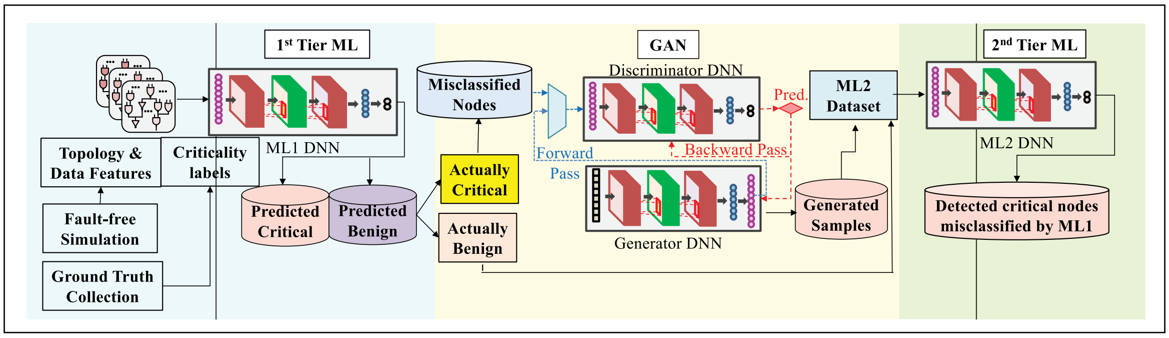


Figure 12. ML-based method for criticality assessment [98].

architecture have demonstrated that hidden layers can tolerate defects even for high defect rates. However, depending on the affected bits or neurons, there may be single defects that can influence the inference accuracy. Defects in hidden layers can be silenced out by a retraining operation with the hardware in-the-loop even for high defect rates. In contrast, the output layer is a defect-sensitive layer, and defects in this layer cannot be masked by retraining necessitating a dedicated fault tolerance scheme.

In [100], defect simulations and Monte Carlo analysis taking into consideration the technology process design kit (PDK) are performed for a spiking neuron. The different faulty behaviors are collected and grouped so as to generate an abstract behavioral-level fault model for spiking neurons that captures the effects of low-level faults, that is, transistor-level defects and process variations. Faulty behaviors turn out to be either catastrophic (i.e., dead neurons that are silenced even in the presence of input activity, saturated neurons that fire nonstop even in the absence of input activity, neurons with a stuck output, etc.) or parametric (i.e., timing variations of the output spike train such as variations in the time-to-first-spike and firing rate). This bottom-up behavioral-level fault modeling approach starting from transistor-level simulations can help generate fault models at a higher abstraction level to be used for software fault injection, while still capturing the effect of underlying root-cause transistor-level faults on the neuron's output, independent of its actual hardware implementation. For example, dead and saturated neuron behavior can be modeled in the output spike train, while timing variations can be modeled by varying various neuron parameters, for instance the neuron's membrane potential threshold.

Chip level

Software-based fault injection is fast and flexible but it ignores the behavior of the AI hardware accelerator. RTL-level, microarchitectural-level, gate-level, and transistor-level fault injection takes into consideration the hardware, but it is slow and inevitably limited to specific hardware blocks.

FPGA-based hardware accelerators offer the possibility to perform realistic fault injection, including faults that would be difficult to model with software simulation, for instance faults affecting the configuration memory or controlling modules. They also offer the possibility to evaluate both accuracy degradation and system exceptions, such as system stall and running overtime. Fault injection experiments on accelerators implemented on FPGAs are presented in [101], [102], [103], [104], [105], [106], [107], and [108]. In particular, fault injection experiments are performed on the FPGA-based FINN quantized neural network (QNN) accelerator [56] in [101], [103], [104], and [105], for the tinyTPU implemented on an FPGA in [107], for FPGA implementations of custom ANN accelerators in [102] and [106], and on neuromorphic FPGA-based hardware supporting SNNs in [108].

In [109] and [110], the soft error reliability of CNN models running on microprocessors is investigated, analyzing the results for different components of the microprocessor and precision bitwidth configurations.

Radiation experiments

Experiences from radiation experiments on different GPUs running different DNN models are described in [81], [82], [83], and [84]. In [81], FIT rates are scaled to a natural terrestrial environment. Main observations are as follows: 1) crashes are more frequent than SDCs but are less critical as they can at least be detected; 2) all reported SDC rates are higher than the 10 FIT limit imposed by the ISO 26262 safety standard for automotive, thereby the reliability of GPU-based AI accelerators is paramount; 3) FIT is dependent on the technology, that is, for FinFET, it is an order of magnitude lower than that of standard CMOS; and 4) error correction code (ECC) protection alone is insufficient to ensure high reliability. In [82], the FIT rate is evaluated for different data precisions, showing that it increases with precision since it depends not only on the fault propagation probability, but also on the probability of the fault occurrence. In [83], the run-time of the inference is tuned based on the beam flux such that the chip experiences no more than a single bit flip event during each application run. The study shows that with the ECC/parity checking enabled, single-bit errors are corrected, no SDCs are observed, and the most stringent ASIL D requirement imposed by ISO 26262 is met. However, vulnerability to permanent faults is observed, which shows that ECC/parity checking must be complemented with periodic structural tests.

Accelerated radiation testing results for DNNs running on FPGAs are reported in [85], [111], [112], and [113]. In [85], it is shown that applying selective triple modular redundancy (TMR) to only the most vulnerable layers can mask a high percentage of faults. In [111] and [112], it is evaluated how reducing the bit-width used for data representation impacts the radiation sensitivity and failure rate. In [113], it is shown that QNNs trained with fault-aware training are more resilient to soft errors.

Finally, results on the reliability of Google Coral tensor processing unit (TPU) to neutrons are reported in [114], considering elementary operations and several CNN models. It turns out that, despite the high error rate, most neutron-induced errors only slightly modify the convolution output and do not change the detection or classification of CNNs.

Memristor crossbar-based architectures

The work in [115] studies the fault injection effect in memristor crossbars. The fault model includes stuck-at faults in the conductance of memristors after programming. The conductance error is defined as the difference between the final programmed value and the target value. A device with a conductance error higher than a positive threshold is considered to have a stuck-on fault, that is, it freezes at a high conductance state. Whereas a device with a conductance error below a negative threshold is considered to have a stuck-off fault, that is, it freezes in a low conductance state. Fault injection experiments show that inference accuracy drops by more than 50% for a stuck-at memristor rate of 20%.

The work in [116] proposes a fault model for SNNs using memristor crossbars for the connection of the layers. Spike timing-dependent plasticity (STDP) is used for learning. Some specificities of the SNN design are that the output neurons are implemented with lateral inhibition, and synapses are off when there is no activity on their connected neurons. The fault taxonomy is divided into different synapse faults and neuron faults. Synapse faults include dead synapses, degraded plasticity, and synapse stuck-at faults. Neuron faults include dead neurons and delayed spiking, as well as faults specific to this SNN design, that is, stuck-at or delayed lateral inhibition and delayed synapse activation fault. Fault injection experiments considered only the worst-case faults, that is, dead neurons and dead synapse faults. Results show that a high fault density is required for a noticeable decrease in recognition rate. Moreover, for dead neuron faults, learning on a faulty network is more critical than a fault occurring in a fault-free trained network.

In [117], the susceptibility of ReRAM-based crossbar arrays to single-event and cumulative radiation damage is investigated. Simulations are performed using an experimentally derived memristor SPICE model. Results for an ANN trained with the MNIST data set indicate that the system is highly resistant to transient single-event effects (SEEs), thanks to the low cross section of the memristive device. Moreover, the cumulative ionizing dose level corresponding to the inference failure point is very large, and thus it is concluded that ReRAM-based accelerators have high radiation tolerance in normal environments.

In [118], it is proposed to train an ML classifier to predict fault criticality in a DNN mapped to memristor crossbars. The considered fault types are stuck-on and stuck-off conductance in the memristor cell. The training set is generated by: 1) random fault injection for which the overwhelming majority of analyzed faults will be benign and 2) a misclassification-driven training (MDT) algorithm to quickly identify critical faults so as to have a balanced training set. The MDT algorithm runs an optimization where the DNN parameters, that is, weights, are perturbed toward maximizing the prediction error. In each iteration, the most significant parameter based on gradient value is chosen. A fault is injected in this parameter and is identified as a critical fault if all samples in a batch of the data set are mispredicted. The features from the benign and critical faults used to train the ML classifier are: 1) fault location; 2) fault type; 3) parameter significance; and 4) parameter deviation amount. The fault criticality analysis can be used to develop a fault tolerance solution that targets only critical faults, thus leading to a significant reduction in the redundancy needed for fault tolerance. The proposed criticality-aware fault-tolerant scheme used in this work is to introduce spare columns for remapping only columns in the memristor crossbar that include cells with critical faults.

General observations from fault injection experiments

Some common conclusions in the above fault injection experiments are as follows.

1) The fault impact depends on the DNN topology, type of layer, and type of activation function used. Moreover, fault sensitivity across layers and across neurons within a layer can vary by several

orders of magnitude. Typically, the output layer is a highly sensitive layer necessitating a dedicated fault-tolerance scheme. Convolution and FC layers tend to spread the SDCs, while subsampling layers tend to mask a significant portion of SDCs. Moreover, very frequently, a bimodal behavior is encountered: either the accuracy is negligibly impacted by the fault, or the accuracy drops rapidly even approaching random guessing.

2) FIT rates of AI hardware accelerators can exceed safety standards, which shows that reliability and error recovery are of paramount concerns.
3) The accuracy drop is contingent on the data set, that is, the application. The same fault can be benign for one data set but can be critical for another.
4) Fault susceptibility depends on the data type used. DNNs using data types of higher dynamic range are more vulnerable. Still, even QNNs with 2-bit precision are shown to be vulnerable. Susceptibility also depends on the affected bit position, with the most significant bits (MSBs) being the most critical.
5) Stuck-at-1 faults furnish the largest accuracy drop because typically over 99% of model parameters have zeros in their MSBs.
6) For systolic array-based accelerators, by increasing the number of layers or the number of neurons per layer, the accuracy drop escalates [127]. This is due to the reuse of the systolic array across multiple layers.
7) For memristor crossbar-based architectures, single memristor yield and endurance are very low, necessitating yield-rescuing methods.
8) For SNNs, saturation neuron faults seem to be the most lethal, although dead neuron faults can also cause significant accuracy drops.

Testability

Introduction

The goal of testability in AI hardware accelerators is no different from traditional hardware: achieving acceptable test quality under manageable cost. It is confronted by the same problems as in the traditional test but with new challenges. While some challenges can be handled by existing tools and solutions, many still remain major problems in today's DFT applications. Some typical issues seen in the industry are discussed below.

Being domain-specific, AI hardware accelerators usually have some unique features that may not be test-friendly. The most prominent one is the sea-of-core design, for example, 1,472 cores in Graphcore GC200 [128], 128×128 systolic array in Google's TPU [129], or even more such as 850 K cores in CS-2 [130]. While the notion of "core" in different accelerators may be very different in size, cores in one design are usually identical or very similar. From DFT and physical design perspectives, these cores may be too small to implement DFT on a per-core basis with reasonable overhead. On the other hand, incorporating many cores in a physical partition could lead to prohibitive costs for DFT or physical implementation and verification, while not taking the advantage of the similarities among cores. Attempting to achieve the best test quality with reasonable overhead, there has been plenty of research on low-cost testing of systems with identical cores [131], [132], [133]. However, for today's AI architectures, an optimal solution might be further explored from other angles, for instance the function structure may help increase DFT test coverage [134], and function patterns may become part of test patterns [135].

AI applications are memory-intensive, hence many AI hardware accelerators require embedded memories with much larger sizes than in traditional ASIC designs. Several megabytes are common practices, for instance 900 MB in Graphcore GC200 [128]. While these memories can be extensively tested and repaired using today's built-in self-test (MBIST) tools, they can present major penalties to power, performance, and area (PPA) [136]. Recently, it has been proposed to bring the computation near to the memory or into the memory, or using large external memory such as high bandwidth memory (HBM) or wafer-bonding. These solutions bring in new challenges for testing. For instance, in-memory solutions may require understanding and creation of new logic and physical fault models [137], while wafer-bonding necessitates better solutions for test access, test power control, and yield improvement [138].

Besides the issues mentioned above, existing test challenges for traditional ASIC design may also become increasingly intense in AI applications. For example, current large AI hardware accelerators require hierarchical DFT solutions that are scalable with design size. However, since an AI hardware accelerator is often a heterogeneous system, hierarchical DFT needs a comprehensive solution for automatic DFT insertion, verification, debugging, and silicon bring up. Another popular difficulty stems from physical design, that is, large accelerators often use a tile-based design where no dedicated routing channels are reserved for global routing. However, complex DFT designs can create hundreds of global signals for scan, MBIST, debug, and so on. This presents a huge overhead for top-level implementation and verification. New solutions are necessary for both efficient DFT and easy physical implementations [136], [139].

AI hardware accelerators are also facing pressure from new marketing and technical trends. For example, as one of today's popular applications, automotive-grade AI hardware accelerators require more stringent screening than before to ensure zero defective parts per million (DPPM), which has to be reflected in the architectural level of DFT design. Test data analysis and diagnosis are also critical for yield and reliability learning [140]. Another typical new paradigm is the 3-D IC design, since large AI hardware accelerators are often limited by physical geometry and cost. Chiplet designs based on 2.5-D or 3-D methodologies can mitigate the challenges from die size, process, cost, and so on, but necessitate a complete set of test solutions from die level, stack level, to package level, which is being addressed in the development of new tools and test flows [141], [142], [143], [144].

Table 2 categorizes some of these challenges or issues (left) and possible solutions (right), which are discussed in this article. This taxonomy is not meant to be comprehensive, but can be representative of many testability activities in AI hardware designs. Table 2 cites only works specific to AI hardware, but more generic solutions will also be discussed, especially on the test architecture side, that are applicable to AI hardware too.

Table 2. New testability challenges for AI hardware accelerators and possible solutions.

Challenges	Solutions
DFT efficiency for designs with large number of cores	Test architectures
Physical design issues for large die size (i.e., routing and timing)	Physical-aware DFT [136]
DFT overhead	Function-aware DFT [134], [135], [145]
Complete DFT solutions for large heterogeneous systems	Functional test generation [127], [146]–[156]
Memory-hungry designs	On-line test [127], [146]–[151], [157]–[166]
New market demands (i.e., 0 DPPM for automotive, on-line test)	Yield improvement [115], [167]–[170]
New architectures and faults (i.e., in-memory computing)	Fault modeling and testing of memristor-based memory technology [171]–[178]
New design paradigms (i.e., 2.5D/3D ICs)	2.5D/3D IC DFT
	Automotive-grade DFT

DFT and ATPG

Test architecture

Plenty of research has been conducted on DFT solutions for identical cores to minimize test overhead and maintain test quality. The assumptions of these techniques may still be valid for AI hardware accelerators, but new solutions may be needed to handle large AI designs with limited cost.

A straightforward idea is to broadcast the test stimuli to identical cores, while comparing their test responses for pass/fail. In [131], a test access mechanism (TAM) is designed to implement an on-chip comparison of multiple identical cores. It contains multiple stages of pipelines and several configurations so that each core's test can be implemented in different modes. As such, it not only supports comparing test responses from cores for manufacturing tests, but also provides diagnosability of a core during silicon bring up and yield ramp up. With on-chip comparison, test data volume can be significantly reduced and so is test time. However, for large AI designs with many cores, such broadcasting-style solutions may encounter increasing difficulty from routing and top integration. It also needs some manipulations of test patterns from standard ATPG tools.

In [132], another form of TAM is proposed supporting similar features. It is a generalized time-multiplexed TAM, where the compressed test and control data streams are serialized before going into the decompressor. At the core level, designers can still utilize the regular decompressor/compactor scan architecture. This simplifies the core-level scan channel configuration and decouples it from top-level scan pin assignment. The designers can be flexible in architecting a core-level scan scheme without worrying too much about the top level. This can be a major benefit for large AI hardware accelerator designs with many cores but very limited scan pin resources. Pattern retargeting, verification, and diagnosis flow are also supported, which are also critical for today's large AI hardware designs.

It can be seen that to take the advantage of sea-of-core design style in AI hardware, on-chip processing of test data may be preferred when implementation is feasible. There are several varieties in this domain. In [133], a TAM design is presented for chips with multiple isolated identical cores. The proposed pipelined architecture relies on forming nonlinear equations on a very limited number of output pins that compress the outputs from the identical cores and solve them off-chip to reproduce the failure information of each core. It uses test resources similar to testing a single core and also supports accurate failure diagnosis. In [178], a TAM based on a majority comparison is presented. It also utilizes an on-chip comparator, yet not to compare with an expected value but with other core's test data to determine a majority value. For example, if more than half of the cores in the comparison present the same value, this value is a majority value. This value is then compared with ATE data. The test cost is close to that of a single core.

In [179], yield improvement is considered when a multicore system contains spare cores. A comparison-based TAM that is capable of handling multiple spare cores is proposed. All faulty cores can be identified via low-cost comparison, and if the spare cores are more than the faulty cores, the chip is still usable. Using spare cores is also common practice in accelerators with many cores. Such designs usually provide a configuration with all good cores and several "partial-good" configurations with different numbers or locations of good cores for yield improvement.

It can be seen that for such DFT solutions to be acceptable, several issues have to be resolved. First, we should be able to manipulate the patterns according to the scan architecture, for example, retargeting the core-level patterns to the top level without regenerating the pattern. Second, test quality such as coverage should not be compromised, and overhead should be minimized. And finally, from an engineering perspective, some critical metrics such as single-core diagnosability, verification effort, and routing complexity should also be considered. Fortunately, some of these requirements are already supported well by current mainstream DFT tools.

Physical-aware DFT

The above works are mostly developed for traditional multicore designs. As discussed earlier, many AI hardware accelerator designs share certain features that may render these traditional solutions either impractical or not as efficient, especially in physical design. To address these AI-specific problems, some new industrial efforts are reported to make DFT solutions more physical-friendly and hence more practical.

In [136], a comprehensive set of DFT solutions targeting AI hardware accelerators are proposed. In the scan test, this work identifies that although accelerators may contain many identical cores, these cores are not as big or as complex as cores in a traditional multicore system such as a CPU. A typical accelerator for data center applications may contain thousands of "small" cores, as shown in Figure 13a. At this core level, any DFT insertion may incur a huge PPA penalty, that is, compression logic, wrapper logic, control logic, and routing for DFT signals. It is too small for the DFT overhead to be economical. On the other hand, if we group many small cores together to create a big partition and apply DFT insertion at this

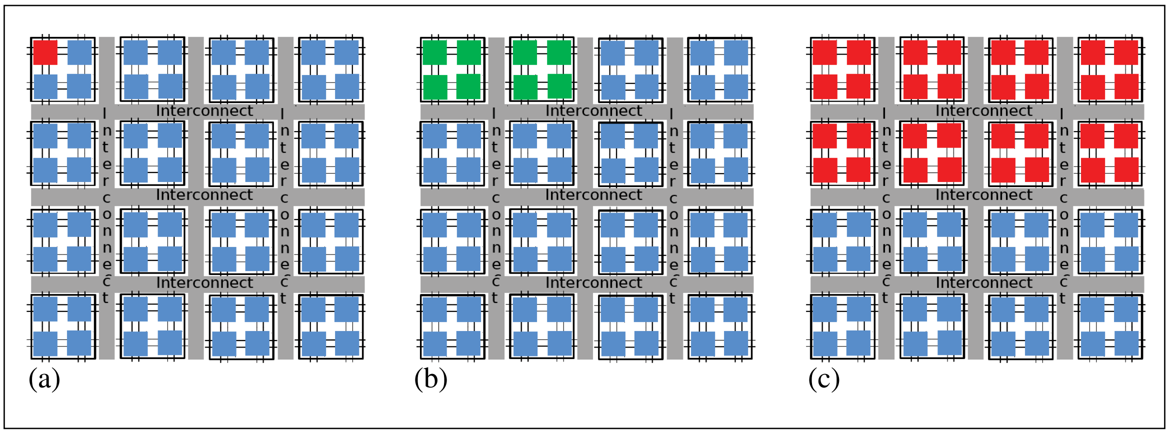


Figure 13. DFT solutions at different levels for many-core AI hardware accelerator designs. (a) Small core level. (b) Core group level. (c) Big partition level.

level, as seen in Figure 13c, the run time, memory requirement, power consumption, pattern count, verification efforts, and other concerns may prevail and render it infeasible, for example, ATPG or simulation cannot finish in a limited time. Meanwhile, the similarities among cores cannot be effectively exploited.

As a result, in [136], it is proposed to find a "sweet spot" where a suitable number of small cores are viewed as a "core group," where DFT insertion, verification, pattern generation, and other activities are done at this level, as shown in Figure 13b. Note that in practice, this usually aligns with physical design requirements, which is probably the most straightforward solution. However, if the physical partition is too big or too small, DFT can still make architectural changes to adapt to a suitable size of the core group. After this core group is determined, existing technologies such as test data broadcasting, test response on-chip comparison, pattern retargeting, and scan channel pin-muxing can be effectively applied.

Note that, in practice, the logically identical cores may not be physically identical. Synthesis and physical implementations may create various physical instances from the same logic module, converting a homogeneous system to a heterogeneous system from a physical perspective. A feasible DFT solution has to take this into account.

A streaming scan network (SSN) [139] is a recently introduced tool that can target these physical challenges. SSN is a bus-based scan data distribution architecture. It contains a scan data bus that travels through all cores in the design, a per-core controller (host) with Internal Joint Test Action Group (IJTAG) support, and regular scan compression logic. The bus is connected to chip-level scan pins, and scan data for any core in the system are streamed in through the bus in the form of packets. The concept of packets is different from that in network switching, since an SSN packet is a fixed-format data segment that only contains scan data, no address or opcode. The local host in each core is preconfigured through IJTAG to learn how to offload scan data from packets. The expected value can also be streamed in for on-chip comparison. The routing and heterogeneous problems with tile-based designs are also mitigated, since only a single test bus is routed through the entire chip. There is no need to pin-mux the scan channels from various cores to top-level scan pins, and the test bus interface is identical for all cores. Another benefit is that due to the flexibility of packeted test data, any cores can be tested at any time. This can help effectively control test power and improve test channel throughput. With comprehensive considerations of DFT and physical-design requirements, this solution is especially suitable for AI hardware containing many identical cores.

Function-aware DFT

Most of the aforementioned technologies are common DFT solutions without an in-depth analysis of the function mode of hardware. Many AI hardware accelerator architectures are domain-specific or even application-specific, hence a customized DFT solution designed for a specific AI architecture is intuitively best for PPA results. To serve this purpose, a DFT architect needs to understand how AI hardware works in a function mode such that the DFT design can be optimized accordingly.

Motaman et al. [135] realize that due to the unique architecture of AI hardware, traditional stuck-at and delay tests may not be sufficient. They study test methodologies and DFT requirements specifically for supervised ML systems. The hardware architecture of FIFO-based and scratchpad-based accelerators is analyzed. Test strategies for specific hardware components such as MAC, global buffer, activation functions, and so on are developed. These solutions are more function-like and can help bridge the gap between traditional test patterns and specific AI hardware test requirements. They are also easy for online tests to ensure product quality.

In [145], postmanufacturing testing of DNN accelerators is discussed. It is argued that the inherent error tolerance can be leveraged to reduce the fault model size and, thereby, the test time and cost. The idea is that if a fault does not lead to inference accuracy degradation for a given accuracy tolerance margin, then it is noncritical and can be dropped. Only critical faults will be targeted during the test application. Two approaches are shown considering a gate-level implementation. The first approach is Boolean satisfiability (SAT)-based structural testing where a SAT solver exhaustively checks all input combinations to determine the fault criticality and generates a test pattern able to detect it. The second approach is classical functional testing where the actual workloads, for example, images, are used as test inputs.

Function mode operation is also studied in [134] to improve test quality in accelerators with very large number of small cores. As suggested in [136], this scenario can be handled by grouping small cores into a core with a size suitable for both DFT and physical implementations. However, it does not exploit the similarity among small cores and test quality may still be impacted if there are interactions between small cores. In typical AI hardware accelerators, there is heavy data traffic between adjacent cores, hence the test coverage on core boundaries is essential. The work in [134] studies intercore connectivity, function dataflow, and design homogeneity to derive a C-testable method that can run ATPG for only a single core to reduce test cost and maintain coverage and diagnosability. By exploring design space, it also presents a hierarchical compaction scheme for on-chip response compaction under reasonable design constraints.

Figure 14 shows a systolic array example consisting of 16 small cores (or PEs). Dataflow is only from left to right and from top to bottom. Each PE has registers on the input sides but not on the output side. If ATPG is performed at each PE level, coverage will be unacceptable since PE itself is not well wrapped by registers. However, if the ATPG run consists of the five adjacent PEs inside the red line, faults in the green PE in the middle will be fully covered. Note that this ATPG pattern can be used to detect faults in all PEs in the same scenario. Since an architecture with small PEs usually contains a large number of them, the overhead of such a scheme is low.

Figure 15 illustrates the sequence of testing the whole systolic array. ATPG consisting of five neighboring PEs is repeatedly used to detect faults in the dark green PEs with high coverage. Each iteration will cover a different set of PEs. Light green PEs are those already covered by previous runs. As a result, for a large systolic array (e.g., 256 × 256), most PEs can be covered by small ATPG patterns in four test runs. The uncovered PEs on the borders can be fully tested in a top-off run.

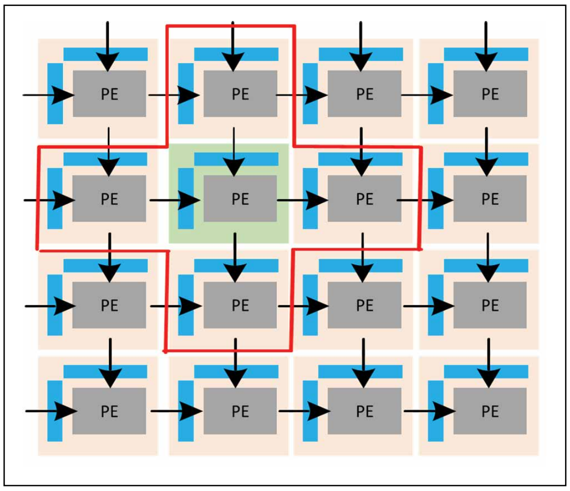


Figure 14. Example of a 2-D unidirectional pipelined dataflow in 4 × 4 PE-based systolic array [134].

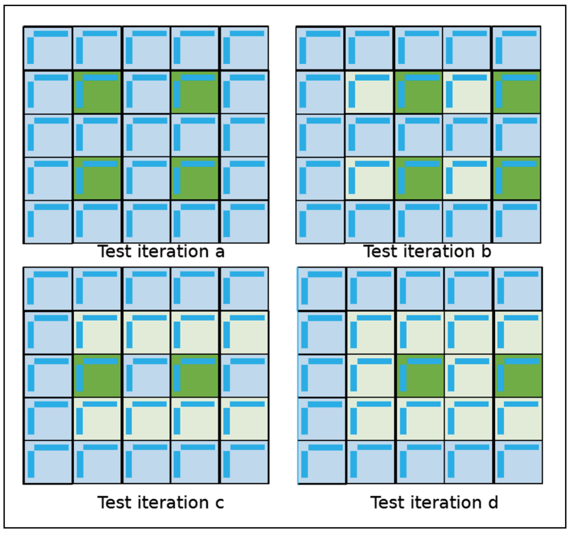


Figure 15. Testing 5 × 5 PE-based systolic array in a checkboard style [134].

Functional test generation

Functional test generation aims at generating inputs, for example, images, that are capable of sensitizing the fault and propagating its effect to the output, leading to a different prediction with respect to that of the nominal fault-free network. This approach has been demonstrated for ANNs [127], [146], [147], [148], [149], including memristive crossbar array-based architectures [146], [147], [149], and for SNNs [150], [151]. As shown in Figure 16, functional tests could be original images from training and testing sets, adversarial examples generated from original images, or synthetic images generated from original images.

More specifically, starting from the available set of input samples, one approach is to select samples that are profoundly similar to other samples belonging to different output classes, for instance a similarity metric could be average pixel intensity [127]. A second approach is to select samples that have been predicted correctly but with the least confidence score [127], [151]. A third approach is to select samples that require more neural network parameter tuning effort during training, where the effort is measured with the change in the loss function in each training step [149]. In [146] and [150], it is proposed to generate adversarial input samples, that is, perturb available input samples by adding a minimum amount of noise aiming at forcing the predictions of the nominal and faulty network to differ. Another strategy is to craft new samples by attaching watermarks to available input samples [147]. The network is deliberately trained to output a designated classification label for a watermarked input, a technique that is called backdooring. Using the watermarked inputs as a validation set, a low validation accuracy indicates a fault. In general, in all aforementioned works, the objective is to use samples that are more vulnerable to misclassification when faults occur. Finally, in [148], a method is proposed for querying a network with a set of specially crafted test inputs, to reveal if the model parameters stored in the memory are faulty.

Functional tests can also be employed by the user of the programmed AI hardware accelerator to validate that the embedded DNN model has not undergone any malicious perturbations [152].

A related research direction is generating error-inducing corner test cases for a trained DNN, which thereafter can be used to retrain the DNN and improve its accuracy [153], [154]. These corner test cases are synthetic real-world input images resulting from realistic transformations of seed images and generated in a way such that they activate a large percentage of neurons in the DNN. For example, for DNN models controlling the perception of autonomous cars, these transformations include changing brightness, changing contrast, shearing, rotation, blurring, fog effect, rain effect, and so on.

In [155], an alternative functional test generation is proposed, demonstrated for memristive crossbar-array architectures targeting detection of classification accuracy drop due to process variability. This approach is inspired by the alternate analog circuit testing paradigm [180], [181]. First, a compact test set of input images is generated with the maximum possible diversity of responses, and a feature vector is defined at the output of the network. An outlier detector in the form of a one-class classifier is trained in the space of features using as training set instances of the DNN with process variations but with acceptable

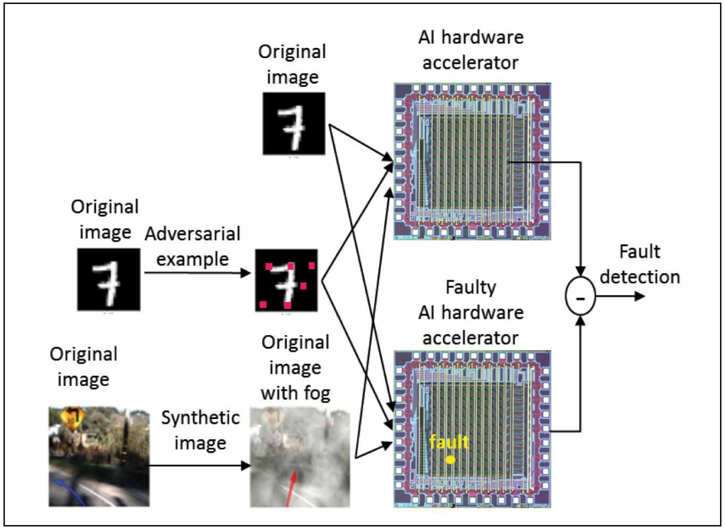


Figure 16. Functional test generation. The street images are from [154]. The chip image is from [60].

accuracy. Applying the compact test set, the outlier detector serves as the first screening of nonconforming devices. Devices that pass this test are presented to a regressor that is trained using the same training set to map the features to the DNN classification accuracy. A guard-band is defined around the minimum tolerated accuracy to classify devices as passing, failing, or fuzzy, where the fuzzy devices fall within the guard-band and are subject to standard testing using the complete image set to obtain a precise decision. Failing and fuzzy devices found in production testing can be combined in batches with prior training data to retrain the outlier detector and regressor.

In [156], a functional BIST scheme is proposed for biologically inspired spiking neurons. The idea is to test that the neuron is capable of producing all the basic firing patterns, that is, regular spiking (RS), fast spiking (FS), intrinsic bursting (IB), and chattering (CH). The test stimulus is composed of low-resolution ramps applied at the bias nodes of the neuron such that in one pass, all firing patterns appear. If one or more firing patterns are missing, then the neuron is declared to be faulty. Examples of functional and faulty neuron responses are illustrated in Figure 17.

Online test

ATPG and functional testing

Many AI accelerators are used in datacenter applications, where reliability, availability, and

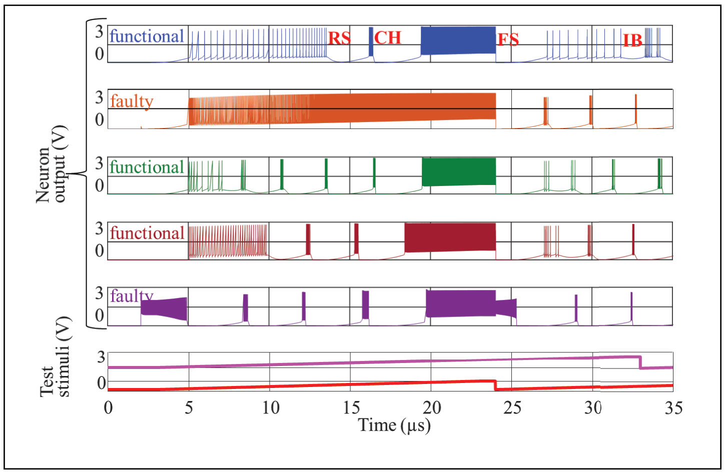


Figure 17. Functional testing of a biologically inspired spiking neuron.

serviceability requirements demand a certain level of online test of memories and logic. However, such tests involve both function mode and test mode, imposing more difficulties on DFT designs.

A recent effort is reported by Amazon AWS in [157], where high-speed serdes I/Os in an AI hardware accelerator are used to transport scan test patterns to test the processing cores. Test patterns are converted to a format compliant with the corresponding protocol and transported from peripheral component interconnect express (PCIe)/universal serial bus (USB), through standard advanced extensible interface (AXI) fabric, to cores. Cores under test will be idled from workload and isolated from the rest of the logic. Although the major advantage of this solution is test time reduction, it supports native online test capability, which is critical in a cloud scenario.

Returning to the functional test generation methods in [127], [146], [147], [148], [149], [150], and [151] discussed in the previous section, as the resultant functional test set is compact, it can also be fed periodically during mission mode in idle times toward functional safety (FuSa).

In [158], different self-test approaches are proposed for the compute units and control units of an accelerator. For compute units that do not contain complex sequential logic, test patterns based on combinational ATPG are generated. For the control units that contain finite-state machines and sequential logic, it is proposed to use functional tests in the form of executing DNN layers with carefully crafted input and weight values. The methodology is enhanced in [159] to cover both stuck-at and delay fault models for both unit types.

Checksums and error codes

The idea here is to build invariants into the accelerator that hold true only in fault-free operations while they are violated in the presence of faults. Thus, checking them concurrently with the operation can point to abnormal operations. Invariants can be built using checksums or error codes.

In [160], a sanity-check mechanism is proposed, in which error detection checksums are constructed by utilizing the linearity property of DNN MAC operations. These linear algorithmic checksums are added to the convolutional layers and FC layers of DNN models after the training. A hardware-based solution is proposed for integration into DNN inference accelerators aiming at reducing the performance overhead at the cost of a minor area and power overhead.

In [161], additional penalty terms, called a balanced checksum, are introduced into DNN training. The balance checksum aims at forcing the DNN layer outputs to adhere to a linear invariant. By adding the balanced checksum into the cost function, error-checking invariants are embedded in DNN model computations. These invariants provide the computation error detection capability during the DNN inference phase, assuming the error would lead to the violation of the trained equilibrium. Furthermore, the introduced custom regularization terms even help a better generalization during the training.

In [162], several algorithm-based error detection (ABED) techniques are presented focusing on the verification of convolution operation, one of the most resource-demanding operations in CNNs. Three variants of ABED are presented to use checksums for filters only, input feature maps only, or both filter and input feature maps. Implementation complexity, runtime overhead, resilience, and performance tradeoffs are studied and compared for the three ABED techniques. This work also addresses the overflow challenges of the checksum arithmetic induced by reduced-precision fixed-point operations (e.g., 8-bit integers). Resilience improvements are evaluated using analytical models, error injection experiments, as well as accelerated radiation experiments.

In [163], an AN code-based fault detection mechanism is proposed to protect the MAC units of the

DNN accelerator. AN codes add redundancy in the data to detect faults during arithmetic operations.

In the case where the weights of the DNN model are loaded encrypted in the memory, an online test scheme is proposed in [164] that employs the padding bits. Padding is used to add a number of bytes to the plaintext to reach a multiple of 16 bytes (i.e., 128 bits) since the encryption is performed on 128-bit blocks. According to the most popular standard, if *n* bytes are added to pad the plaintext, then each of the bytes will encode the value *n*. If a bit-flip occurs, the on-chip decryption module will spread it creating multiple bit-flips affecting also the padding bits. The fault detection scheme is then to check that the decrypted padding bytes indeed encode the value *n*. Using this approach, most single faults become critical, but they become detectable at the same time.

Software based

In [165], online test strategies based on Software Test Libraries (STLs) are proposed for embedded systems running ANN applications. STL is composed of selftest routines that are executed during boot-time or run-time. The strategies are categorized into two groups according to whether they incur or not a small penalty in the inference time. Zero-penalty strategies include: 1) run part of the STL during weight data transfer when the PEs are idle and 2) test the inactive PEs of a low-intensive computation layer and cover all PEs in subsequent inferences using a scheduler based on a round-robin algorithm. Small-penalty strategies include: 1) one PE is executing a self-test while the rest of the PEs share the AI workload; 2) apply the entire STL between two consecutive inferences; and 3) arrange and apply the entire STL between successive layer computations when weight data transfer is happening. The strategies are evaluated on CNNs running on an open-source RISC-V platform. First, STL is verified to have high stuck-at test coverage. Then, the different strategies are evaluated based on the inference time penalty and fault detection time (FDT) tradeoff, where FDT is the worst-case time to detect a fault from the moment of occurrence.

Memristor crossbar arrays

An online concurrent fault detection method for memristor crossbar arrays is proposed in [166]. The underlying observation is that faults affect dynamic power consumption. An indirect simplified measure of the dynamic power consumption is used, in particular, the number of logic "1s" at the outputs of the ADCs digitizing the output current of the crossbar's columns. An adder-tree design is used to count the number of "1s," which incurs a small area overhead. The time series corresponding to the power consumption and count of "1s" show a strong correlation when faults are present, which allows using the count of "1s" as a simplified metric. When abrupt changes occur in the time-series data, the presence of faults is indicated. Changepoints are detected by examining time series within a sliding window. For a current time point, the sliding window is centered on it. The probability density functions of the points in the left- and right-hand segments are estimated and compared to examine if the current time point is a changepoint. When a changepoint is detected, the percentage of faulty cells in the crossbar is estimated. A regression model is trained for this purpose offline. A variety of independent feature variables are used, including statistics of the time-series data, average weight stored in the crossbar, and average input applied to the crossbar's input. Error correction is invoked when a high percentage of faults is estimated. For example, the faulty crossbar can be replaced with a redundant crossbar.

Yield improvement

In [167], the yield and accuracy-aware optimum test of AI accelerators (YAOTAs) framework is proposed. The framework deals with stuck-at faults in MAC units of AI accelerators and considers output bit position *K* up to which the inference error is acceptable. Faults in the fan-in logic cones of bit positions lower than *K* are considered noncritical, while fan-in logic cones of bit positions higher than *K* are considered critical. ATPG test patterns are applied and if only noncritical faults are found in the MAC, then the PE may be acceptable depending on how many such faults exist, the AI workload, and the error tolerance limits demanded by the application. If critical faults are found, then the PE is permanently disabled. The map of faulty PE locations is programed in a fault status register. For single instruction, multiple data (SMID) architectures where PEs are interconnected with NoC/mesh, PEs can be individually switched off and bypassed. For systolic array-based accelerators, a deactivation protocol is proposed without hardware-level modifications. In particular, it is proposed to deactivate the PE columns that contain PEs with critical faults and shift input data by inserting dummy rows of zeros. This approach has no area overhead but decreases the execution throughput. By adopting this framework, the manufacturer can avoid discarding the full accelerator chip because of the presence of a few faulty PEs, thereby increasing yield.

In [115], two methods are proposed to recover the fabrication yield loss of memristor crossbar-based accelerators due to high memristor defect rates. The first method consists of identifying memristors that are stuck at certain conductance levels and performing retraining of the network where only defect-free memristors are adjustable. In the case where the performance loss cannot be fully compensated by retraining, the second method presents a remapping algorithm where memristor columns that are heavily polluted, that is, contain many defective memristors, are replaced by additional redundant columns.

In [168], spatial redundancy-based fault-tolerant schemes are proposed for yield loss recovery of memristor crossbars. The fault model considers stuck-at fault in memristor cells, that is, a memristor cell can be stuck-at a high resistance state or low resistance state. The fault-tolerant schemes apply to designs where the dot-product operation is mapped to two memristor crossbars. In particular, once the model is trained, the mapping allocates the positive weights to a "positive" crossbar and the negative weights to a "negative" crossbar. The proposed fault-tolerant mapping algorithm is to make the positive and negative weights eliminate the impact of faults on each other. For example, if a positive cell is stuck-at, the weight of the negative cell is enlarged accordingly to approximate the target weight. This approach works if only one of two cells in the same location in the two crossbars is faulty at a time. Spatial redundancy schemes are proposed in the case where the fault rate is high, where the same concept of pairwise fault elimination is used. These schemes make use of redundant crossbars, crossbar columns, and cells.

In [169], methods are proposed for improving the yield of memristor crossbars in the presence of memristor resistance variations and stuck-at faults. It is assumed that the resistance variations and the location of stuck-at cells can be detected. Two different methods are proposed in the case of multilayer perceptions (MLPs) and CNNs. For MLPs, the problem of mapping the weight matrix of the trained model to the conductance matrix of the crossbar is formulated as a bipartite matching problem. The metric used is the summed weighted variations across the cells. Then, in a second step, the derived new weight matrix is fed as a starting solution to an off-device training algorithm. The algorithm aims at iteratively reducing the weight with the maximal deviation. This is done by scaling down the weight in each training epoch and adapting the weights of the surrounding cells to recover the classification accuracy. For CNNs, the method exploits the fact that two memristor crossbars are used to represent positive and negative weights since the conductance of a cell can only be positive. The weight is expressed as the difference between the two conductances. Therefore, there is a bitwise redundancy in the architecture. The proposed method is to reprogram the resistance of one cell of the pair to eliminate the resistance variation in each cell. The same principle is used as a self-compensating mechanism to tolerate stuck-at faults. In the second step, off-device training and on-device training with few iterations so as to consider the limited endurance of the memristors can be performed to improve the classification accuracy.

A common technique to improve the error-resilience of DNN accelerators is to extract the memory fault map using postmanufacturing testing and perform fault-aware retraining of the model. Doing so for each faulty chip results in significant retraining overhead. In [170], it is proposed to train many faulty chips at a time. The fault maps of chips are merged into a unified fault map, which is then used for retraining a single model that will be loaded to every chip. A fault map is abstracted as a 2-D table where an element corresponds to a memory cell. The state of the cell is encoded to 1 or 0 for an S-A-1 or S-A-0 fault, respectively. For contradictory locations where S-A-1 of one fault map overlaps with S-A-0 of another fault map, the policy is to select the polarity that incurs less accuracy drop in the DNN inference. The retraining speedup increases with the number of fault maps merged. However, this speedup is at the expense of an accuracy drop compared to per-chip retraining. Empirically, it was observed that exposing the DNN gradually to faults rather than exposing it to all faults from the very start allows the DNN to learn at a faster rate and achieve better accuracy.

Fault modeling and testing of memristor-based memory technology

Memristors offer a compelling solution to the scalability problem of AI hardware accelerators, as they can be used as nanoscale synapses. They offer also a promising in-memory computing architecture that solves the data transfer bottleneck as discussed in a previous section. Besides these applications in an AI hardware context, memristor-based memory technology has a large potential for replacing traditional memory technologies and is the focus of today's research. As memristors are susceptible to process, voltage, and temperature (PVT) variations and manufacturing defects because they are fabricated with new materials and processes, there is a large body of work that aims at understanding such failure mechanisms and accurately model them to develop optimal postmanufacturing memory tests, including march test algorithms and DFT [171], [172], [173], [174], [175], [176], [177], [178]. These works find applicability in the context of AI hardware accelerators implemented with memristive crossbar arrays.

Dependability

Introduction

AI hardware is generally integrated into some intelligent or autonomous systems required to operate throughout their life cycle in a highly dependable manner. The dependability issues of AI hardware have aroused great interest in recent years. Dependability is a broad term used to define the ability of a system to deliver its intended service [183]. Any system, including AI hardware, can be viewed as a group of components integrated into one single entity to serve the purpose of delivering a certain service (e.g., AI algorithm acceleration for an AI accelerator). Throughout the life cycle of system deployment, there may be a service failure triggered by intrinsic or extrinsic effects, whereby the delivered service deviates from the intended one. The dependability of a system is the ability to avoid such service failures that are beyond the acceptance level.

Dependability encompasses a broad spectrum of attributes, which are quantities to measure dependability from various perspectives. The main attributes include reliability, availability, maintainability, and safety [183], [184].

- Reliability, availability, and maintainability are three highly related attributes, which are usually measured by statistical metrics. Reliability denotes the continuity of the correct service. The level of reliability is commonly specified in terms of mean time to failure (MTTF) [184]. Maintainability denotes the ability to repair when a service failure has occurred. It can also be specified as a statistical term with the mean time to repair (MTTR) metric which represents the expected system down time (including repair time) [184]. Lastly, availability denotes readiness for correct service. It can be expressed as a function of MTTF and MTTR as A = MTTF/[(MTTF+MTTR)] [184].
- Safety denotes the ability of a system to not cause harm to people, things, or the environment. Safety includes FuSa and safety of the intended functionality (SOTIF). FuSa is defined as the absence of unreasonable risk due to hazards caused by malfunctions [185], [186], [187]. On the other hand, SOTIF focuses on the absence of risks caused by performance limitations of the intended behaviors or by reasonably foreseeable misuse by the user [188].

We note that security is often not characterized as a single attribute of dependability. While highly related to dependability, security is considered a composite notion combining confidentiality, availability, and integrity attributes [184].

From a dependability perspective, there are various threats leading to a potential violation of the targeted goal. A threat at the component operation layer is usually called a fault. There are two main categories of faults: intrinsic faults and extrinsic ones. The former may be originated from aging effects, device variability, latent manufacturing defects, and susceptibility to environmental conditions (e.g., radiation causing soft errors, electrical/mechanical stress). On the other hand, there are some faults caused by system inputs (e.g., malicious inputs to AI systems, or user misuse), which are said to be extrinsic or external.

There are a variety of techniques to improve the dependability of a system, including fault prevention, removal, tolerance, and prediction. Fault tolerance is one of the most popular means aiming at tolerating a fault in a functional system. There are different levels of fault tolerance requirements, for example, fail-operational, fail-safe, and so on. A fail-operational system upholds the continued functionality and intended services in the presence of a fault. There are two main subcategories: 1) upholding service without performance degradation and 2) with degraded performance. The latter is commonly referred to as fail-degraded or fail-reduced. A fail-safe system aims at transitioning the system to a well-defined condition to maintain a safe state in the event of faults. The FuSa mechanism is one example of means to achieve a fail-safe property. Note that there is another term “fail-silent,” which is described as the guarantee of no service (e.g., no system output) in the event of failures. Such a silent state can be viewed as a specific defined safe state, thus from this perspective, fail-silent can be considered as a subcategory of fail-safe.

As AI hardware provides service to more and more mission-critical or safety-critical applications, these hardware elements need to be evaluated for compliance with the dependability goal (e.g., safety). In general, they share the same dependability theory foundation and requirements as other hardware (e.g., traditional general-purpose processors). However, there are several novel dependability challenges as well as opportunities introduced by unique characteristics of AI hardware computing architecture, application, and also research and development cultures.

First, AI hardware goes under a new computing paradigm called “domain-specific computing” [189]. Domain-specific computer architecture with domain-specific hardware acceleration has been introduced in recent years to address performance needs that general-purpose computing is hard to meet. This emerging computing paradigm shift is expected to bring new opportunities to AI hardware dependability method development. For example, while many traditional application (domain)-agnostic fault tolerance techniques, for example, ECC or TMR, are commonly used in general-purpose computing, an alternative technique with exploiting domain-specific characteristics of AI hardware could be pursued to achieve better efficiency in terms of PPA.

As discussed before, many research works suggest that DNNs have inherent resilience to moderate variations of parameters and activations. Such approximate nature of DNNs enables the development of approximate computing to support efficient AI learning in resource-constrained hardware, especially for inference. However, the actual impact of AI hardware faults could be more severe on AI application service results, for example, classification accuracy. It demands thorough hardware fault analysis and novel lightweight fault-tolerant techniques exploiting the architectural properties of AI hardware.

We also note that some research suggests deep-learning models may have an inherent weakness against input perturbation, for example, adversarial examples. The adversarial robustness of DNNs has received particular attention, and there is a rapidly growing body of research work in this field [190], [191]. Such adversarial inputs can be viewed as external/extrinsic faults. In this survey article, we focus on dependability against intrinsic faults, that is, those induced by AI hardware internally while potentially stressed by environmental effects or workloads. Other security threats that will not be covered in this survey include DNN model IP theft [192], [193], [194], backdoor attacks on DNNs performed when training is outsourced [195], [196], and fault injection attacks [197], [198].

Another characteristic of AI hardware is that it is often an integral part of some AI-based solutions consisting of multiple interacting system layers—from hardware/physical to software/application. From this perspective, AI hardware dependability strategies should use a system-based approach beyond the techniques limited to local hardware. The concept of cross-layer dependability or cross-layer resilience [199], which leverages the inherent fault-tolerance of multiple layers, should be used for AI hardware to exploit domain-specific faults at the system level. Moreover, heterogeneous computing containing AI accelerators along with general-purpose CPUs and/or FPGAs has gained mainstream adoption in the computing industry [200]. There exist far greater opportunities for exploiting heterogeneity to achieve system-level dependability.

Deep-learning-based AI has become a revolutionary tool in many industry fields, with seemingly unlimited potential to outclass traditional techniques. This is a burgeoning field filled with opportunities as well as chaos, much like the new kind of “Wild West.” We see that industry and academia are eager to push out AI innovations, with new architectures and higher performance expressed mainly with the tera operations per second (TOPS) metric, so as to battle for technical leadership in this rapidly growing field. In general, the AI field is permeated by a pioneering and risk-taking spirit. On the other hand, conservatism is fundamental in the dependability field (especially for safety). It is in sharp contrast with the pioneering spirit and self-regulation philosophy. A paradigm shift is needed to bridge the gap between them. Over the past few years, there have been growing efforts in this direction. For example, Europe has started legislation to make the use of AI safer and more ethical, such as in critical infrastructure impacting people's lives and health [201]. Still there is a considerable gap between the AI dependability goal and the available solutions. This is a research frontier where the technical community can contribute more to bridge the gap by introducing new methods.

Design-for-dependability aims at enhancing the reliability, availability, maintainability, and safety features of the AI hardware accelerator. All these attributes boil down to rendering the AI hardware accelerator error-resilient. We classify the existing design-for-dependability approaches into four categories, as illustrated in Figure 18. The first category includes model-based approaches where the goal is to derive a model that meets the performance requirements and additionally it has intrinsically built-in or programmed error-resilient capabilities such that by construction when mapped onto hardware, it is capable of tolerating certain hardware-level faults. The second category includes proactive hardware-based techniques where the goal is to make the accelerator design passively tolerate certain hardware-level faults. The third category includes reactive hardware-based techniques where the goal is to make the accelerator react to an occurring fault in realtime, including built-in monitoring of fault occurrence and low-latency error recovery whenever a fault has occurred. The final fourth category includes cross-layer approaches where the error tolerance objective is shared between the model and hardware.

A taxonomy of existing techniques under the different categories is provided in Table 3. These techniques will be presented in more detail next.

Model-based approaches

Figure 19 combines and illustrates model-based approaches that will be discussed next in detail.

Model training modification

A number of works propose to achieve fault tolerance by modifying training. The first method is to add artificial faults and noise into the network during training such that the network learns to tolerate faults [61], [202], [203]. A second method is to restrict weights

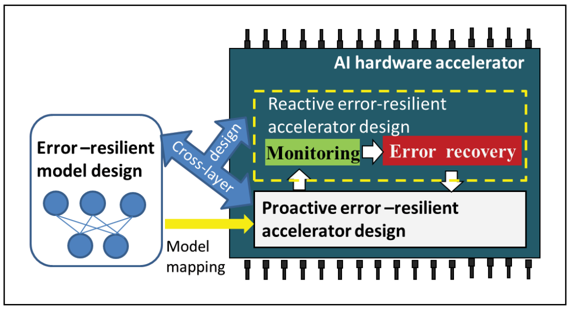

AI hardware accelerator
Reactive error-resilient accelerator design
Monitoring
Error recovery
Proactive error–resilient accelerator design
Error–resilient model design
Cross-layer design
Model mapping

Figure 18. Design-for-dependability approaches.

to have low values since intuitively fault-tolerance degrades by the use of large values [202]. A third method is to add a penalty term to the training cost function that takes into account errors that arise due to faults and multiply the penalty with a regularization parameter that controls the tradeoff between the degree of fault tolerance and inference accuracy. The underlying idea is to bias the solution toward a fault-tolerant network. Approaches in this category include constraining the weights to lie within a limited range toward an even weight distribution [204], [205], [206]. A fourth method is to combine the training process and fault-tolerance objective into an optimization problem solved by nonlinear optimization algorithms with the aim to learn a network model that performs the desired task and at the same time fulfills fault-tolerance constraints [207], [208], [209], [210]. A fifth method proposed in [211] considers constructive training in the presence of faults, where neurons are incrementally added whenever the network fails to learn until a satisfactory learning or a user-defined maximum network size is reached.

The aforementioned approaches are early works targeting shallow FC networks and considering faults at the behavioral level. They laid the foundation of several approaches for modern AI hardware accelerators presented recently which are discussed next. A thorough and comprehensive review of these early approaches is provided in [212].

Fault-aware training

In [213] and [91], it is demonstrated for ANNs and SNNs, respectively, that training with dropout improves error resilience. Dropout was originally proposed in [276] to prevent overfitting and reduce the generalization error on unseen data. The idea is to temporarily remove neurons during

Table 3. Taxonomy of design-for-dependability approaches.

Model-based	Proactive hardware-based	Reactive hardware-based	Cross-layer
Model training modification [61], [90], [91], [202]–[223]	Memory cell re-design [224], [225]	Weight-shifting [226]	Model/hardware co-design [227], [228]
Model modification [229]–[231]	Memory aging mitigation [232]	Re-learning [233]	Fault-aware pruning with re-training [95]
Fault-tolerant model search [234]	Activation clipping [57], [81], [235]–[238]	Algorithmic-based fault-tolerance [81], [239]–[244]	Fault-aware mapping [101], [245]–[249]
	Redundancy-based [77], [85], [91], [101], [104], [250]–[256]	Fault masking [91], [92], [257]–[261]	Variation-aware mapping for memristor crossbar arrays [262], [263]
	ECC [81], [83], [84], [264]	ML-based [265]	Adaptive training after testing [146], [266]
	Razor [257], [267], [268]	Neuron adaptation [269]	Aging-aware on-line training of memristor crossbar arrays [270], [271]
	Hardening against radiation [272]		Thermal-aware optimization of memristor crossbar arrays [273]–[275]

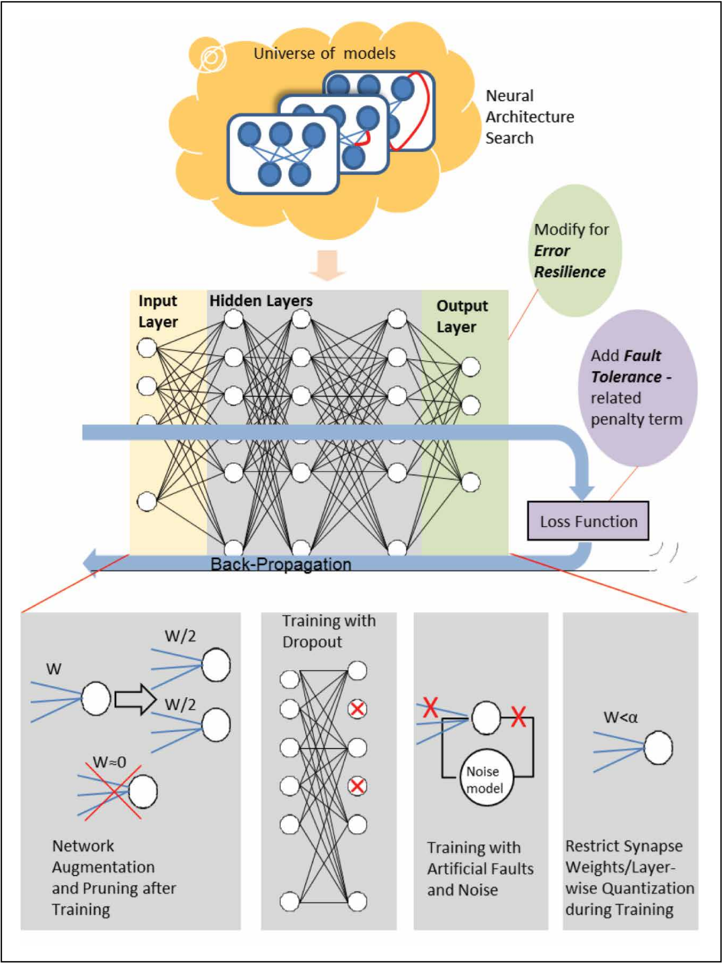

Universe of models
Neural Architecture Search
Modify for Error Resilience
Input Layer
Hidden Layers
Output Layer
Add Fault Tolerance - related penalty term
Loss Function
Back-Propagation
W
W/2
W/2
W≈0
Network Augmentation and Pruning after Training
Training with Dropout
Noise model
Training with Artificial Faults and Noise
W<α
Restrict Synapse Weights/Layer-wise Quantization during Training

Figure 19. Model-based approaches.

training with some probability *p*, along with their incoming and outgoing connections. At test time, the final outgoing synapse weights of a neuron are multiplied by *p*. For a network with *n* neurons, there are 2^n "thinned" scaled-down networks, and training with dropout combines exponentially many thinned network models. The motivation is that model combination nearly always improves performance, and dropout achieves this efficiently in one training session. The reason why dropout is a natural fault-aware training approach is that it equalizes the importance of neurons across the network, resulting in more uniform and sparse activity across the network. Therefore, if a neuron becomes faulty, this turns out to have no effect on the overall inference accuracy. In [91], it is demonstrated that training the SNN with dropout can nullify the effect of dead neuron faults and neuron timing variations in all hidden layers, while the SNN can withstand a multiple-fault scenario with high dead neuron rates. A technique equivalent to dropout, called erasure regularization, is to set neuron activations and weights to zero during training [214].

In [215], an error injection layer is developed that allows injecting faults according to a fault model during training time. The FINN FPGA-based QNN accelerator for CNNs [56] is adopted for the study. The focus is on two main fault types for CNNs, namely single-channel stuck-at faults and the same pixel in all channels stuck-at. Training is performed on a GPU and fault injection on FPGA. Results show that this fault-aware training approach: 1) improves the error-free accuracy by behaving like a regularizer; 2) leads to highly fault-tolerant networks with accuracy very close to the error-free one; and 3) offers an improved hardware cost versus worst-case accuracy tradeoff when selective TMR is used to compensate errors in the most critical layers.

Another fault-aware training approach is to inject bit errors in the weights during the training process. This strategy has been investigated in [214], [216], and [217] showing that it allows a margin for voltage reduction in the memory of the DNN accelerator, thereby helping to reduce energy consumption. In other words, the accuracy drop due to bit errors resulting from voltage under-scaling can be compensated by this fault-aware training approach.

Training with noise

In [218], [219], and [220], it is shown for memristor crossbar-based architectures that injecting noise during software training enhances the robustness of inference to the nonideal effects of memristor crossbars. In [218], a Gaussian noise source is incorporated at the crossbar outputs, while in [219] and [220], a random noise term is injected into the weights during training.

Co-optimizing inference accuracy and fault tolerance

Techniques to unify inference accuracy maximization and fault-tolerant improvement optimization are proposed in [90], [221], and [222]. In [90], a variant of an evolutionary optimization-based training algorithm for SNNs is proposed where the fitness function is redesigned aiming at improving the error-resilience capability. In particular, the fitness function becomes a weighted sum of the baseline accuracy and the average accuracy obtained on a faulty version of the network when imposing a certain synapse fault rate. In [221], process variations and noise are modeled as random variables and are incorporated into the weights of the neural network during training. In [222], a framework is presented that utilizes a Bayesian neural network to conduct variation- and defect-aware training. The approaches in [221] and [222] are demonstrated for memristor crossbar-based architectures.

Restricting numerical ranges

The range of parameters inside each layer of a DNN can vary a lot. This can be a major source of vulnerability to bit errors in DNNs. For example, considering a conventional fixed-point data format, the variation in the first few MSBs can be very detrimental to small parameter values. In [219], it is proposed to use the dynamical fixed-point (DFP) data representation formation that allows to adaptively change the location of the decimal point based on the range of data. In particular, by left shifting the decimal point position, we can make sure that there are no unused MSBs. In [223], to reduce the vulnerability surface, layer-wise quantization techniques are proposed to tighten the quantization margins to match the utilized range in each DNN layer. Also, a new regularization method, called outlier regularization, is introduced in the training phase to further tighten the numerical range and shape the parameter distributions.

Model modification

In [229], it is proposed to augment the trained network by replicating critical neurons and their associated connections. A neuron and its replica have half the weights of the original neuron to maintain the network mapping. The underlying idea is that if a critical neuron fails, then the effect on the inference will be lower, thanks to spatial redundancy.

In [230], it is proposed to prune unimportant nodes in the network according to a sensitivity analysis and then retrain the pruned network. Redundant nodes are also introduced so as to share the task of critical nodes.

In [231], a method is proposed to enhance the error resilience of DNNs by modifying just the output layer that performs the binary classification. Typically, an ensemble of independent logistic classifiers is used, each implementing a winner-takes-all rule by one-hot encoding. Error-correcting output code (ECOC) learning is applied to optimize the coding matrix and increase the Hamming distance of codewords assigned to different classes. This work proposes a collaborative logistic classifier extended from the logistic classifier to ease neuron competition and improve the error capacity. Increasing the decision distance on final classification is shown to rectify the accuracy degradation induced by faults across the complete architecture. The method is cost-effective, scales to any network size, and can be easily integrated with existing hardware-level fault-tolerant techniques.

Fault-tolerant model search

In [234], a neural architecture search (NAS) algorithm, such as the one proposed in [277], is employed to discover a fault-tolerant architecture. The employed NAS algorithm uses reinforcement learning rewarding architectures toward maximizing performance. In this work, the NAS algorithm is modified to add a second term in the reward that expresses fault tolerance to bit flips (FT-NAS).

Another version of the algorithm computes the first term of the reward, that is, the classification accuracy, by inducing faults during training (FTT-NAS). The hand-designed networks show performance degradation already with a very small bit-flip rate. Instead, the network found by FT-NAS shows a graceful degradation with an increasing error rate, whereas the network found by FTT-NAS achieves near baseline accuracy for a high error rate. The discovered fault-tolerant architectures are inspected and they are found to establish double connections between some pairs of nodes. In other words, sensitive connections are identified by the algorithm and redundant paths are added for defending against faults.

Proactive hardware-based approaches

Traditional fault-tolerant methods continue to play critical roles in AI hardware. For example, ECC is used to protect the memories of AI hardware accelerators [81], [83], [84], [264]. The Razor technique [278], aiming at detecting and correcting circuit timing errors, is also used in some AI hardware accelerator designs [257], [267], [268]. Besides these standard domain-agnostic fault-tolerant techniques, there are other different proactive hardware-based approaches for AI hardware, illustrated in Figure 20, that will be described in more detail in this section.

Memory cell redesign

In [224], a passive fault-tolerant method for ReRAM-based crossbars is proposed by redesigning the memory cell to have a 2-transistor/2-resistor (2T2R) structure, where each bit of information is stored in a differential fashion. In particular, the pair low resistive state (LRS)/high resistive state (HRS) means logic value zero, while the pair HRS/LRS means logic value one. Readout is performed by comparing the resistance values of the two differential devices, thus doubling the memory read window with regard to the conventional 1T1R cell architecture shown in Figure 5. This differential architecture reduces the number of bit errors due to device variations and limited endurance. Its benefits are demonstrated on a BNN. This inherent fault-tolerant architecture has auxiliary advantages. Weak programming conditions can be applied to achieve energy savings. It also features outstanding

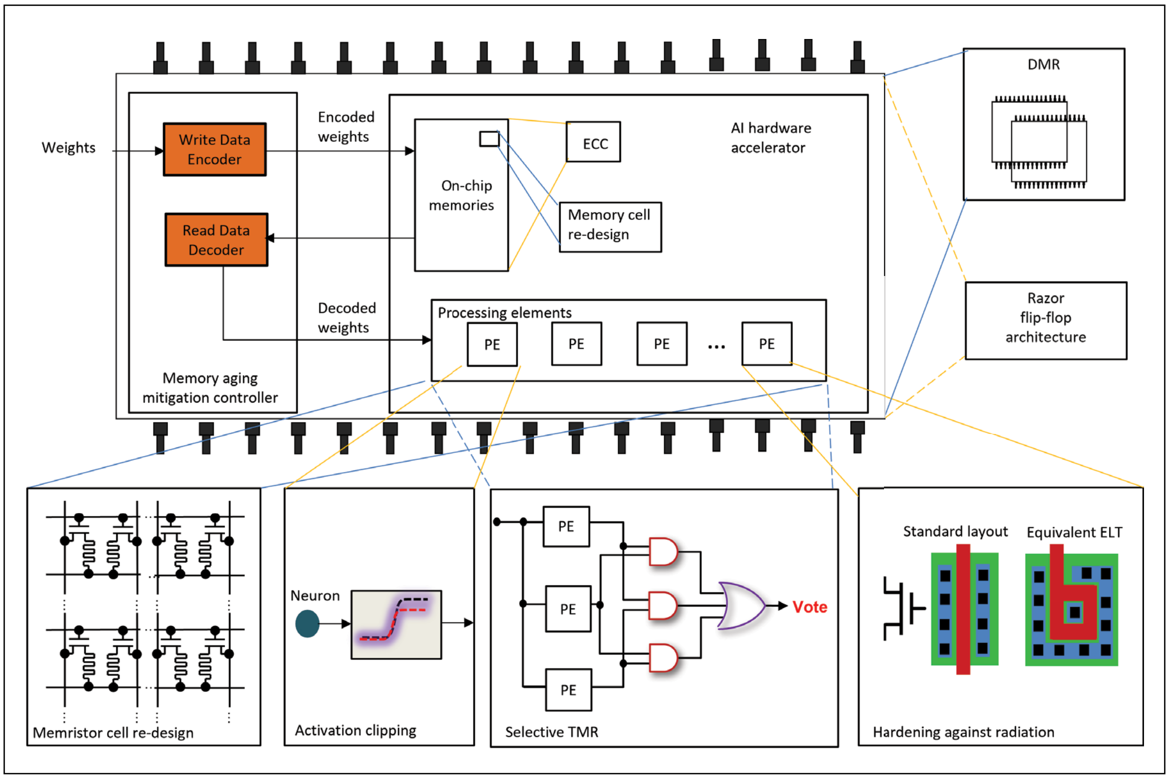


Figure 20. Proactive hardware-based approaches.

endurance opening the way to the possibility of on-chip training of neural networks.

In [225], a hardened SRAM cell is proposed for DNN accelerators. Based on the key observation of sparsity in DNNs (i.e., weights have a strong bias toward zero) and given that bit flipping from zero to one is more likely to cause a failure of DNN outputs, the proposed memory cell provides robust immunity against node upsets and reduces the leakage current dramatically when zero is stored in the cell.

Memory aging mitigation

A low-overhead aging mitigation scheme of weight memory buffers in DNN accelerators is proposed in [232]. The underlying observation is that optimized aging can be achieved by balancing the duty cycle of the memory. To this end, a microarchitecture is proposed composed of a write data encoder (WDE) for encoding the weights before writing them to the on-chip memory, and a read data decoder (RDD) which performs the inverse function when reading the data from the on-chip memory and before passing it to the PEs. The WDE XORes the incoming weights with a common 1-bit enable signal that is generated by a true random bit generator (TRBG), thus adding a sense of randomness to the bits to be written in the memory. The output of the TRBG is periodically inverted by XORing it with a bitstring stored in a register to account for the scenario where the TRBG is biased toward either “0” or “1.” The RDD performs the same XOR operation as the WDE on the outgoing bits. Results show that this scheme offers maximum aging mitigation for any data representation and across different accelerators and DNN models.

Activation clipping

In [235], it is observed that as the fault rate increases, the activation of neurons becomes more intense. In [57], when the activation output of a neuron exceeds by 10% the expected range of values, it is considered a symptom of an error occurring. To this end, in [235], it is proposed to use a clipped version of the activation function such that when activation exceeds a threshold, then the neuron is silenced. A search algorithm using the area under the curve accuracy versus fault rate as a metric is proposed to find the optimal threshold that maximizes classification accuracy under different fault rates. This strategy is investigated also in [236], [237], and [238]. In [236], values are truncated to the maximum value observed in the training set. While in [235] and [236], activation functions are bounded globally per layer, in [237], the truncation value is fine-grained per neuron. In [238], to avoid the risk of false positives, it is proposed to compute several single statistics on neurons’ output values, that is, minimum, maximum, average, and standard deviation. If at least two different statistics are out of range, then a fault detection is flagged.

In [81], it is proposed to redesign the maxpool layer of CNNs so as to halter the fault propagation. The redesign consists in evaluating if the value of the max element is higher than a threshold and, if so, then halt the processing of the frame and move on to the next frame, or use the second largest element if it is reasonably small.

Redundancy-based

State-of-the-art AI hardware accelerators for autonomous driving vehicles employ dual-modular redundancy (DMR) to ensure safety for the system [250], [251], which requires substantial hardware resources.

One idea is to perform selective TMR applied to the most critical layers instead of a full TMR, which is inspired by the observation that different layers have different sensitivity to faults [85], [91], [101], [104]. Selective TMR is feasible resource-wise, and the resultant area and power consumption overhead can be tolerated. Typically, the most critical output layer is protected with TMR, which is enough to achieve a high level of fault masking. For deep networks, the output layer accounts for a small percentage of neurons of the whole network, thus the percentage overhead of applying TMR only to the output layer scales down.

Redundancy-based fault tolerance can also be applied at different hierarchy levels, for example, TMR of critical kernels [77], DMR of critical feature maps [252], TMR of MSBs in computational blocks such as adders and multipliers [253], and TMR of critical neurons [254].

In [255], a redundancy-based fault-tolerance strategy, called hybrid computing architecture (HyCA), is proposed for the 2-D array of PEs that greatly reduces the overhead of the classical DMR. The basic idea is to add a separate set of dot-production processing units (DPPUs) in parallel to the original computing array of PEs. HyCA can be utilized to scan the entire

2-D array and detect the faulty PEs at runtime, and recompute all the operations that are mapped to the faulty PEs, independent of the location of faulty PEs.

Finally, a redundancy-based fault-tolerant strategy based on ensemble learning is proposed in [256]. Ensemble learning consists of training a set of independent smaller and weak (i.e., with lower accuracy) base networks, using different network structures, learning algorithms, and training data sets. Thereafter, the results are combined, for instance using voting or averaging, to improve task performance. The idea is that when one or more weak networks fail due to a fault, the ensemble of other networks can still operate reliably.

Hardening against radiation

In general, ionizing radiation, depending on the energy of the incident particle and the time of exposure, can give rise to transient events or permanent damage, such as bit-flips, shift in the transistor's threshold voltage, and an increase in the leakage current. Transistor hardening refers to applying changes in the layout so as to tolerate exposure to ionizing radiation. In [272], a spiking neuron design is hardened by redesigning the transistors' layout using an enclosed layout transistor (ELT) topology for the gate. This particular neuron uses a memristive device to implement the memory element, that is, the membrane, of the neuron. The area overhead with respect to the original design excluding the memristive device is 4.51x. However, taking into account the memristive device, it is argued that area overhead is negligible because the memristive device is placed on top of the CMOS subsystem during the back-end phase which requires an extensive area.

Reactive hardware-based approaches

Figure 21 illustrates different reactive hardware-based approaches described in this section, separating the two underlying mechanisms, namely fault/error detection and localization and fault/error mitigation.

Weight shifting

In [226], the weight-shifting fault-recovery mechanism is proposed. If an incoming synapse of a neuron is detected faulty, then the loss is compensated by adapting the weights of other synapses. If a neuron is faulty, then its outgoing synapses are treated as faulty.

Relearning

In [233], a high-level biologically inspired model of the cortical structure of the brain is developed capable of performing feed-forward sensory processing and automatic abstraction for visual inputs. The model is trained using Hebbian learning with

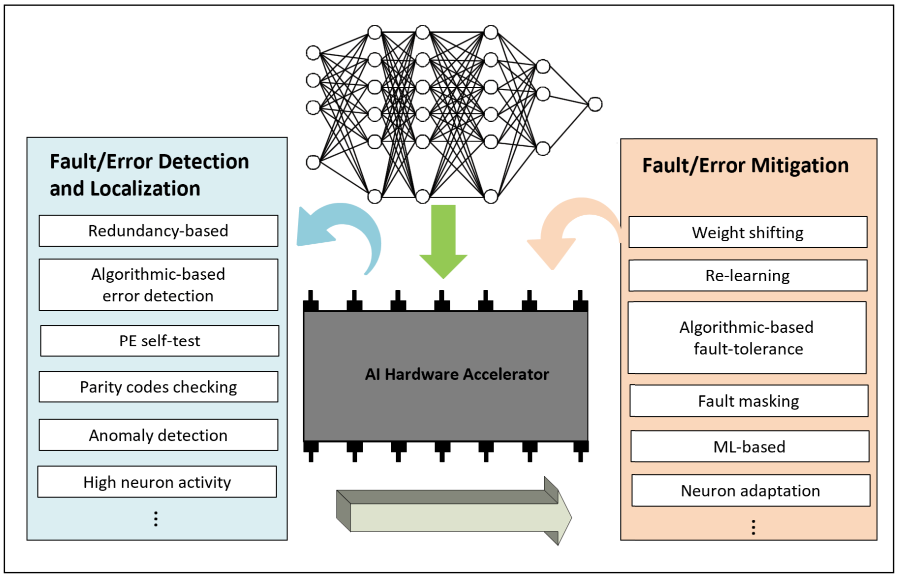


Figure 21. Reactive hardware-based approaches.

repeated exposure to input samples. A software version of the model is deployed on a GPU for fault-tolerant experimentation. The fault model considers neurons stuck-at faults, that is, neurons that do not fire when they should (stuck-at-0) or they fire when they should not (stuck-at-1). Single and multiple fault scenarios are studied including spatially distributed and clustered faults. For stuck-at-0 neurons, the network is capable of relearning as their functionality is taken over by neighboring neurons. On the other hand, stuck-at-1 neurons can severely degrade the performance and upon detection are disabled and the network relearns. Detection is performed by interrupting the operation and recomputing the response of the winning minicolumn of neurons on two neighboring minicolumns. A voting scheme is used to determine a defective minicolumn. This is a form of TMR but uses the existing redundancy. The model's accuracy with relearning shows a graceful degradation to faults and a large number of faults can be tolerated.

Algorithmic-based fault tolerance

Algorithmic-based fault tolerance (ABFT), originally proposed in [279], is a low-cost solution for detecting and correcting abnormal behavior in matrix–matrix multiplications based on checksums. As neural network operation heavily relies on matrix–matrix multiplications, ABFT finds a natural application for enabling fault tolerance in AI hardware accelerators with several ABFT schemes being proposed to date in the literature [81], [239], [240], [241], [242], [243], [244].

As an example, in [242], the compute underutilization of inference-optimized GPUs is exploited by evaluating the computing resource bottleneck for GPU kernels. The metric being used is a comparison between the arithmetic intensity of the kernel (in GPU terminology, a GPU kernel consists of multiple threads that can be executed in parallel) versus the compute-to-memory-bandwidth ratio (CMR) of the GPU. A kernel is compute-bound if the arithmetic intensity is higher than the CMR; otherwise, it is memory-bandwidth-bound. For a memory-bandwidth-bound kernel (i.e., with low arithmetic intensity running on a high CMR hardware), there is an opportunity to leverage the underutilization of compute units to allow ABFT execution on unused resources. Motivated by this observation, a finer-grained ABFT scheme is proposed, referred to as thread-level ABFT, as illustrated in Figure 22. Performing ABFT at the thread level can exploit compute underutilization of bandwidth-bound kernel to reduce the execution time overhead of ABFT. Furthermore, an arithmetic-intensity-guided ABFT is proposed that selects the best ABFT scheme for each individual layer of the network, for example, global-level (i.e., kernel-level) ABFT for compute-bound layers, and thread-level ABFT for memory-bandwidth-bound layers.

An implementation of ABFT for memristor crossbar-array architectures is proposed in [240]. As illustrated in Figure 23, a crossbar of size $r_{\text{xbar}} \times c_{\text{xbar}}$ is partitioned into smaller crossbars of size $r_t \times c_t$. For each smaller crossbar, two extra columns are added. In the first column, the cell in row i computes the nonweighted checksum $G_r(i,\ 1) = \Sigma_{j=1}^{c_t} G(i,j)$, where $G(i,j)$ is the nominal expected conductance value of the cell in position (i,j) of the crossbar. In the second column, the cell in row i computes the weighted checksum $G_r\,(i, 2) = \Sigma_{j=1}^{c_t} W_G(j) \cdot G(i,j)$, where $W_G(j) = j$. For each smaller crossbar, M test input vectors are applied, denoted by $\mathbf{V_t}(k) = [V_t(k,\ 1),\ldots,\ V_t(k,\ r_t)]$, where $V_t(k,\ i) = V_0 \cdot W_t(k,\ i)$, V_0 is a unit voltage, and $W_t(k,i) = (f(i))^{k\text{-}1}$, $f(i) = 2^{i\text{-}1}$, $k = 1,\ \ldots, M$. The outputs of the two checksum columns for test input k are $O_s(k,\ 1) = \Sigma_{i=1}^{r_t} V_t(k,i) \cdot G_r(i,\ 1)$ and $O_s(k,\ 2) = \Sigma_{i=1}^{r_t} V_t(k,i) \cdot G_r(i,\ 2)$. The output of crossbar column j for test input k is $O_t(k,j) = \Sigma_{i=1}^{r_t} V_t(k,\ i) \cdot G'(i,j)$, where $G'(i,j)$ is the actual conductance value of the cell in the (i,j) position of the crossbar. Two signatures are defined for test input k, namely $A(k) = (\Sigma_{j=1}^{c_t} O_t(k,j) - O_s(k,\ 1))/Vo = \Sigma_{j=1}^{c_t} \Sigma_{i=1}^{r_t} W_t(k,i) \cdot [G'(i,j) - G(i,j)]$ and $B(k) = (\Sigma_{j=1}^{c_t} w_G(j)\ O_t(k,j) - O_s(k,\ 2)/V_0 = \Sigma_{j=1}^{c_t} \Sigma_{i=1}^{r_t} w_G(j) \cdot W_t(k,i) \cdot [G'(i,j) - G(i,j)]$. In fault-free operation, $A(k) = B(k) = 0$. Based on the percentage of faulty cells, the size $r_t \times c_t$ is chosen such that no more than two faults occur in a small crossbar. In this case, using $M = 4$ test inputs, we can perform fault localization and compute conductance deviations in faulty cells in both the crossbar and the checksum columns using the two signatures. In particular, we can write eight equations with six unknowns, that is, the fault locations, denoted by $(x_1\ ,y_1)$ and $(x_2,\ y_2)$, and the conductance deviations, denoted by d_1 and d_2, for the two faults, and solve the system of equations with linear algebra.

Fault masking

In [92] and [257], memory bit-flip mitigation schemes are proposed with no costly fault-tolerant

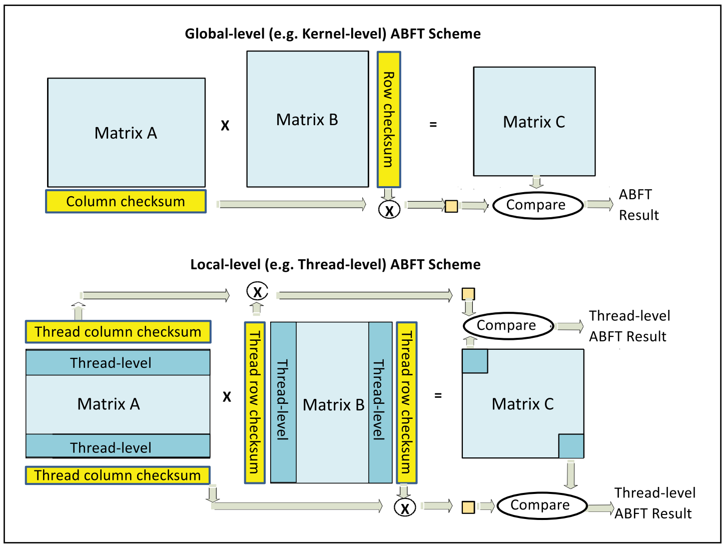


Figure 22. ABFT global and local schemes.

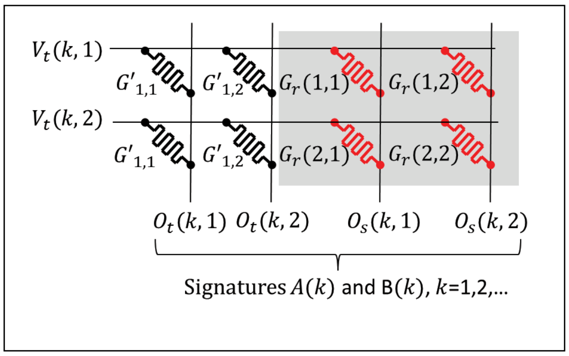


Figure 23. Memristor crossbar checksums.

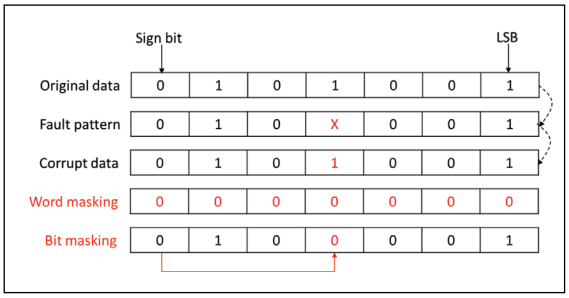

Sign bit
LSB
Original data 0 1 0 1 0 0 1
Fault pattern 0 1 0 X 0 0 1
Corrupt data 0 1 0 1 0 0 1
Word masking 0 0 0 0 0 0 0
Bit masking 0 1 0 0 0 0 1

Figure 24. Word and bit masking error mitigation techniques.

operations relying on the sparsity of data. The assumption made is that information is available on which bits are affected, for example, using Razor shadow latches that can detect faults by monitoring circuit delays [278]. The schemes are based on masking faulty bits. The two main schemes, namely word and bit masking, proposed in the case of fixed-point data representation, are illustrated in Figure 24. Word masking sets all bits of the corrupted register to zero. This is equivalent to setting the synapse weight to zero which intuitively, due to the sparsity of the network, will have a lesser impact on the accuracy as opposed to leaving uncorrected a $0 \rightarrow 1$ bit-flip in a high-order position. Bit masking sets a faulty bit equal to the sign bit and can tolerate more faults than word masking. It achieves a similar effect by rounding the synapse weight toward zero.

In [258], a soft error detection and correction scheme is proposed for CNNs accelerated on FPGAs. Fault injection analysis shows that single-event upsets (SEUs) on PEs are far more consequent than SEUs occurring in memory. Moreover, SEUs in MSBs are shown to be far more critical. It is proposed to execute a self-test of PEs during free cycles motivated by the fact that the average PE utilization ratio is usually below 85% during inference. The self-test consists in exercising the higher bits of multiplexers and adders in the PE separately, and this traversal overhead can be easily confined within the free cycle. Temporary error mitigation is achieved by using a zero setting upon SEU detection, instead of reconfiguring the PE immediately.

In [259], a fault-tolerant design of the systolic output stationary (OS) DNN architecture is proposed. Faults in the data path, that is, outputs of PEs, are detected online and mitigated. A functional online test approach is proposed where neighboring PEs are tested separately by applying the same input (i.e., one PE needs to be taken offline) and checking if their outputs are identical. The fault mitigation approach is to mask the faulty PE's output to zero. As a PE roughly corresponds to a single neuron, performing training with dropout can augment robustness. This fault-tolerant approach shows no latency in the inference and in terms of area overhead it requires the addition of three MUXes per PE and an external comparator for the whole PE array.

In [260], the opportunistic parity (OP) fault mitigation technique is proposed for protecting CNN weights. OP is based on the observation that errors in the LSBs of the weights can be tolerated. The idea is to flip the LSB if needed such that the weight has even parity. Checking the parity code can detect an odd number of bit flips. Noting that a memory word can be large and multiple weights can be stored in one memory word, we can adjust parity for individual weights or the entire memory word. When a parity error is detected, the weight values are replaced with zeros.

Regarding SNNs, with the passive neuron fault tolerance scheme based on dropout in place, active neuron fault tolerance in hidden layers needs only to address neuron saturation (see previous sections) [91]. A compact online monitor can be used per neuron to detect this symptom [91]. The monitor, shown in Figure 25, is based on a small-sized counter that counts the number of spikes a neuron produces after every single input spike and has a reset port connected to the input of the neuron. A saturated neuron will produce spikes with higher frequency than usual, causing the counter to overflow before an incoming spike resets it again. A latch is set when an overflow happens and an error flag is raised. On the other hand, in fault-free operation, the neuron needs to integrate multiple input spikes before it can produce a spike of its own, hence the counter is always reset, and the error flag signal stays at zero. If saturation is detected, the "fault hopping" concept is proposed as a recovery mechanism [91]. The idea is to turn a saturated neuron into a dead neuron since the network can withstand dead neuron faults. This simplifies the hardware implementation requiring adding a single extra transistor per neuron. An example is shown in Figure 26 where a transistor shown in red is added to cut off the biasing of the spiking neuron when the flag signal indicating neuron saturation goes high.

In [261], a run-time soft-error mitigation technique for SNNs is proposed. A fault criticality analysis shows that increased weights and neuron saturation are the only faults that can decrease inference accuracy. For synaptic faults, it is proposed to perform weight bounding. In particular, if the weight is greater than a threshold, then it is replaced with a predefined value (e.g., zero or maximum weight value from the nominal SNN). For neuron saturation faults, if the membrane voltage stays above the threshold for more than two clock cycles, then spike generation is disabled similar to [91].

ML based

In [265], an ML-based method is proposed to detect an anomaly in a DNN and mitigate the effect at run time. The fault model is transient faults in the form of random single bit-flips in the buffer memories and data paths of the accelerator. For a given

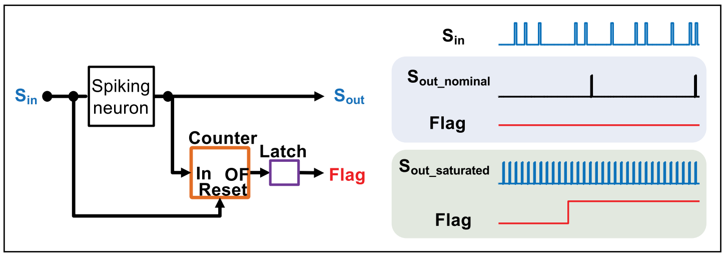


Figure 25. Symptom detector for a spiking neuron.

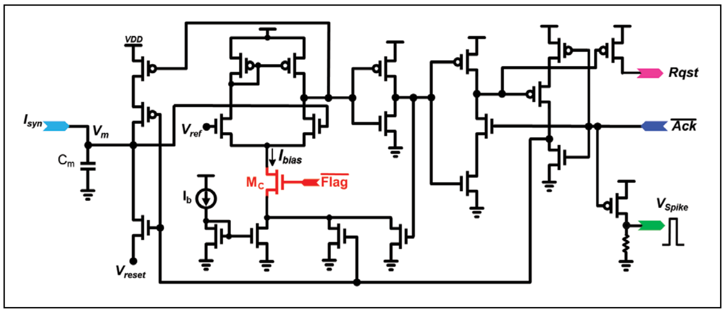


Figure 26. Spiking neuron design with cut-off transistor enabled when the neuron starts saturating raising the flag signal high.

input, each layer of the DNN provides a set of feature activations (i.e., the respective neuron output values). A unified feature activation trace is generated by concatenating the feature activations of all layers. Then, a small FFNN, named as error detection and mitigation network (EDMN), is trained in this feature space to perform anomaly detection due to critical bit-flips, as well as to predict and recover the correct classification result for error mitigation. The training data is generated by random bit-flip injection simulations recording the feature and classification result. Furthermore, the small EDMN can be safeguarded against faults by using classic methods, for example, TMR.

Neuron adaptation

An application-specific fault-tolerant design of an SNN implemented in an FPGA in proposed in [269]. The SNN is used to control the motion of a robotic car, that is, speed and direction, establishing an obstacle avoidance task. There are four motor neurons controlling the forward (F), right (R), left (L), and reverse (REV) movements. The neuron's excitatory synapse receives input current according to the obstacle distance. Prioritization is achieved

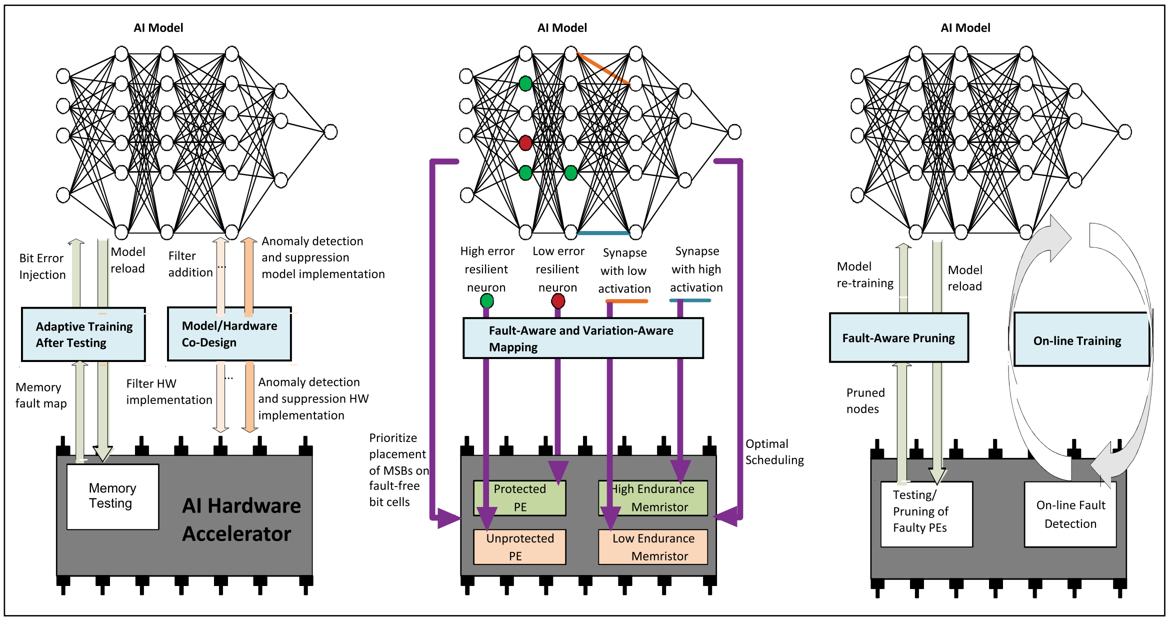

AI Model
Bit Error Injection
Model reload
Filter addition
Anomaly detection and suppression model implementation
Adaptive Training After Testing
Model/Hardware Co-Design
Memory fault map
Filter HW implementation
Anomaly detection and suppression HW implementation
Memory Testing
AI Hardware Accelerator
AI Model
High error resilient neuron
Low error resilient neuron
Synapse with low activation
Synapse with high activation
Fault-Aware and Variation-Aware Mapping
Prioritize placement of MSBs on fault-free bit cells
Optimal Scheduling
Protected PE
High Endurance Memristor
Unprotected PE
Low Endurance Memristor
AI Model
Model re-training
Model reload
Fault-Aware Pruning
On-line Training
Pruned nodes
Testing/ Pruning of Faulty PEs
On-line Fault Detection

Figure 27. Cross-layer hardware-based approaches.

via the inhibitory synapses. The neuron's spiking rate detects the activity of the corresponding motor, that is, F, R, L, or REV. Fault tolerance is achieved by using many synapses instead of one receiving the same input. The neuron monitors the total injected current from all synapses during a time window, and if an abrupt or abnormal variation is noticed, then this points to a fault occurring in one or more synapses. Fault tolerance in this context means retaining the same firing rate. This is achieved with one of two mechanisms: 1) adjust the neuron's threshold and 2) adjust the operating frequency of the neuron. Mechanism 2) is achieved via the dynamic partial reconfiguration feature of the FPGA that provides a way to generate custom clocks on-the-fly. The adjustment scheme is not continuous, but it is based on a lookup table (LUT) for given expected-erroneous pairs of input currents.

Cross-layer approaches

Figure 27 illustrates together different cross-layer hardware-based approaches that will be described in detail in this section.

Model/hardware codesign

In [227], a method is proposed to maintain DNN accuracy under high error rates by suppressing the numerical contributions of anomalous activation. It first integrates anomaly detection and suppression layers into DNN models. To address the training challenges due to the discontinuous nature of these layers, a two-stage training process is proposed to ensure a fast convergence with competitive accuracy. A hardware module is proposed to perform anomaly detection and suppression at the inference phase of the DNN accelerator.

In [228], a median feature selection technique is introduced to alleviate the impact of bit errors before the numerical operation of each layer. It is observed that the critical bit errors are often those leading to a significant numerical increase in the activation or weight magnitude. Such errors exhibit characteristics similar to the spike noise patterns in the image-processing field, where order-statistics filters have been proven to be effective against large spike noises. Therefore, DNN models are first trained with integrated median filters. After achieving the desired accuracy in training, the model is deployed on the AI accelerator with dedicated hardware performing median filtering operations.

Fault-aware pruning with retraining

In [95], a fault-tolerant scheme is proposed for systolic array-based DNN accelerators, depicted in Figure 28. In the first step, the scheme includes fault-aware pruning where the faulty MAC is bypassed using multiplexing, which is equivalent to setting the MAC's weight to zero. In the second step, the pruned systolic array is retrained. It is demonstrated that this fault tolerance scheme can maintain a classification accuracy close to the baseline even when up to half of the MAC units are faulty. It is also demonstrated that with the fault-tolerant scheme in place, more aggressive voltage under-scaling can be employed to provide energy savings while not sacrificing classification accuracy.

Fault-aware mapping

In [245], the underlying hypothesis is that neurons with strong contributions have a high impact on the inference accuracy if they are faulty, thus a strong contribution implies low error resiliency. To derive the ranking, an algorithm is proposed based on the Taylor decomposition of the network and layer-wise propagation of the contribution. The average contribution is considered taking the mean over the training set. Thereafter, it is proposed to design an accelerator to have a number of protected PEs and memory buffers, where protected means that they are safeguarded against faults by utilizing spatial or temporal redundancy and error correction mechanisms. The neurons with the lower error resiliency are mapped to protected elements, whereas neurons with the highest error resiliency are mapped to unprotected and unreliable elements.

In the case of overlay architectures that consist of an array of PEs on which layers or a portion of layers are scheduled to be executed in sequence, whenever a single PE is faulty, this affects multiple outputs both within a layer or among layers. Thus, the portion of the neural network affected by the corrupted PEs depends on the scheduling. One zero-overhead approach, therefore, would be to identify and utilize the optimal scheduling that minimizes the accuracy drop [101].

In [246], first, the faulty PEs are pruned after testing, similar to [95]. Given the saliency of the weights, it is proposed to map neurons of a layer on different segments of the hardware such that the sum of the saliency of the weights that are mapped on pruned PEs during inference is minimized.

In [247], it is proposed to first identify the most critical neurons and then determine an optimal scheduling that distributes evenly the critical neurons to the available PEs such that if a PE exhibits a fault this affects the functionality of a limited number of neurons.

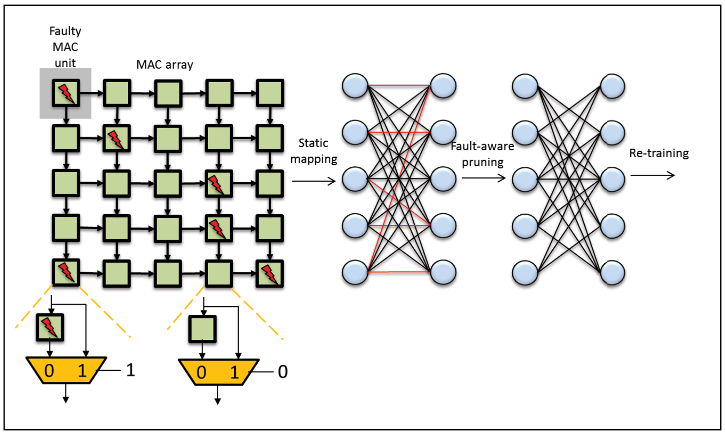


Figure 28. Fault-aware pruning followed by retraining.

In [248], it is proposed to first derive the memory fault map using testing and then apply a fault-aware mapping consisting of bit shuffling to prioritize placing the MSBs on the nonfaulty memory cells. This strategy is also investigated in [249].

Variation-aware mapping for memristor crossbar arrays

Line resistances degrade the voltage levels along the crossbar columns, thereby inducing more errors at the columns away from the drivers. In [262], it is proposed to rank the DNN kernels based on sensitivity analysis and rearrange the columns such that the most sensitive kernels are mapped closer to the drivers.

In [263], it is shown with circuit simulations that a memristor crossbar presents a current imbalance, that is, asymmetry in the current propagating through its different memristors. This current instability is due to the parasitic components on the horizontal and vertical wires of the crossbar that result in voltage drops. For example, the current on the largest path from a presynaptic neuron to a postsynaptic neuron, that is, the path that traverses the top horizontal line through the upper right memristor and down the far right vertical line, is smaller compared to the current on the smallest path, that is, the path that includes only the bottom left memristor. This current variation results in endurance variability of memristors in the crossbar, where endurance is defined as the ratio of average failure time and switching activity. For example, the memristor in the upper right corner will have higher endurance, whereas the memristor in the bottom left corner will have the lowest endurance. Based on this observation, the *eSpine* framework is proposed for endurance-aware mapping of SNNs to neuromorphic hardware. Given the SNN workload, the objective of *eSpine* is to find an intelligent mapping of neurons and synapses to neuromorphic hardware, such that synapses with high activation are implemented on memristors with high endurance and vice versa.

Adaptive training after testing

In [266], a methodology is proposed, named memory adaptive training with in-situ canaries (MATIC), aiming at aggressive voltage scaling of weight SRAMs in AI hardware accelerators obtaining significant energy savings, while maintaining the inference accuracy. The idea is to perform read-after-write and read-after-read operations on each SRAM address of the chip, to generate a profile or failure map of the marginal, failure-prone bit cells. Then memory adaptive training is performed where the profiled bit errors are injected into the training process enabling the DNN to compensate via learning. In this way, during normal operation, by applying voltage scaling, a significant fraction of resultant bit errors is passively tolerated. Furthermore, tunable accuracy-energy tradeoffs can be achieved by using a select set of bit cells that are on the margin of read failure as a canary. The canaries are replicated critical bit cells that can detect imminent failures. Such in-situ canary bits can be polled at run-time to determine whether voltage modifications should be applied to maintain an advantageous accuracy-energy tradeoff.

In [146], periodical online testing is performed using a functional test set of adversarial examples. If the network is found faulty, memory fault diagnosis is performed using a march test. If a soft fault has occurred, the remedy is to refresh the memory with a model backup stored in the edge device. If the fault is permanent, the fault map is sent to the cloud for model retraining after masking the faults, and afterward, the model is retransmitted to the edge device.

Aging-aware online training of memristor crossbar arrays

The writing endurance of memristor cells ranges from 10^6 to 10^8 write operations, whereas the training phase can take 10^5–10^7 iterations. Therefore, online training in memristor crossbar-based accelerators causes degradation of the valid resistance range of the memristor, an effect called aging in the memristor, resulting in most memristor cells becoming faulty. In [270] and [271], frameworks are proposed combining software training and hardware tuning to counter the aging effect. For example, in [270], first, the threshold training method is introduced to reduce the number of write operations in each iteration. The observation is that for the vast majority of weights, the weight update is very small. In this case, the weight update is suppressed. Second, after a fixed number of iterations, a fault detection method is executed to detect stuck-at faults and update the status of cells. The method, called quiescent-voltage comparison, consists of the following steps: 1) divide the crossbar into smaller crossbars; 2) for each crossbar, perform a write operation with the same write change to every cell; and 3) compare the actual crossbar outputs to the expected reference output and if a discrepancy is found, then it means that at least one cell in the selected crossbar is stuck-at and cannot be updated when we write an increment. By using a smaller crossbar size, we increase fault detection accuracy at the expense of higher test time. After fault detection, the third method exploits the fact that over half of the weights are zero. The idea is to map zero weights to cells that have stuck-at-0 faults. This can be achieved by reordering the column and rows of the weight matrix. To do this efficiently while respecting the inherent connection of cascaded layers, it is proposed to reorder only the neurons.

Thermal-aware optimization of memristor crossbar arrays

Temperature increase changes the conductance value of a memristor cell and decreases its endurance. In [273], [274], and [275], thermal-aware training and online optimization schemes are proposed to resolve the temperature-dependent retention issues of memristors with one-time DNN deployment.

Perspectives

Fault criticality assessment

The lesson learned from published fault injection experiments is that not all faults are equal. Most end up being benign, that is, they are masked, their effect on the output is not large enough to result in accuracy loss, or the accuracy loss is insignificant. On the other hand, there are some suspect critical faults, for example, 0→1 bit flips in high-order bit positions, stuck-at-1 artificial neuron activations, saturated spiking neurons, and faults in the last layers, especially in the output layer. Yet, the locations of critical faults cannot be safely presumed and depend on many factors, such as the network architecture (e.g., depth, number of channels, etc.), the sparsity of the network (e.g., percentage of near-zero weights), and the cognitive task (e.g., data set). Given that the fault space explodes for deep networks, to speed up reliability analysis one of the main challenges is reducing safely the fault space aiming at circumventing simulation of faults that would prove to be benign. For example, we can use ML to predict fault criticality and block the simulation unless a fault has some likelihood of being critical. This will allow for avoiding speculative and unguided fault sampling and evaluating more faults, thus identifying with higher probability the critical faults that will need to be dealt with fault tolerance techniques. Identifying the critical faults will allow better targeting the fault tolerance strategies and reducing test costs.

Along this direction, we require faster automated fault injection frameworks. More "tricks" to speed up simulation can be integrated into current frameworks. For example, as every layer is computed sequentially, if a fault is masked in an intermediate layer, then simulation can be stopped early. Or, for a fault in a given layer, we can start the simulation from this layer considering the golden fault-free response of the previous layer.

A second main challenge is developing fault models for higher-level descriptions of the accelerator such that they are plausible in hardware and capture well-foreseen hardware-level faults. Many works consider faults at an abstract behavioral level that do not necessarily map to hardware or their probability of occurrence from a hardware point of view is very small. Most fault injection experiments consider a subset of possible faults or a subset of the subblocks of an accelerator. Fault injection experiments on actual hardware or radiation experiments can shed more light on the impact of faults but ideally, the impact should be assessed earlier at the design stage so as to add the necessary provisions on-chip for fault tolerance.

Testability

As discussed above, a plethora of new testability features and methodologies have been proposed in the literature and adopted in practical AI hardware accelerator designs. However, moving forward, there is still a pressing need for novel DFT solutions to target existing and upcoming challenges. We have seen various demands emerging on the horizon for products in the next several years.

First, as part of ordinary ASIC flow, DFT activities are tightly associated with the design and physical design in many aspects, such as the turn-around cycle, physical design tools, methodologies, and even computing resources. Since many of today's large AI hardware accelerator designs are indeed challenging the design and physical design limitations, posing direct threats to project delivery, successful DFT solutions should recognize and attempt to help mitigate these threats. For example, large AI hardware accelerator designs usually cannot be readily fit into existing computing resources (i.e., servers, emulators, etc.), hence design needs to be sliced into multiple modes and multiple partitions. Without a feasible solution, the number of modes and partitions may quickly get out of control and computing resources will soon be depleted. In many cases, DFT verification may demand even more resources than function verification. Existing tools provide basic help, yet a major function and DFT verification framework still have to be handcrafted to ensure resource availability and design space coverage. On the physical design side, many issues have been addressed in earlier sections and current DFT tools have had a strong focus on solving these problems. However, major challenges continue to bother the DFT owners, such as timing budget, PPA request, signal routability, performance correlation, and so on, not only in large AI hardware accelerator designs for cloud utilization, but also in smaller designs used in edge devices, which are extremely sensitive to power and area. As such, DFT architects have to be well versed in all stages of the design flow and ensure their customized DFT architecture and flow can accommodate the design and physical design requirements to meet the target of PPA and time-to-market.

Second, new AI hardware accelerator architectures may require the advances of novel DFT architectures, algorithms, or models, needless to mention the exotic neuromorphic and in-memory computing hardware, which have spurred research on many new DFT architectures and fault models. Even the traditional style of design may encounter a much higher level of challenges than before. For example, as mentioned earlier, DFT for large AI hardware accelerator designs needs to be multimode and multipartition in design and verification spaces. This may incur an excessive number of test patterns, causing ATE memory overflow or unaccepted test expenses. This is not a new problem, but exacerbated in AI hardware accelerators. Since there are no readily adopted solutions, DFT architects need to be creative in designing some on-chip hardware for a low-cost test. On the other side, problems rarely seen before may emerge as the new norm. For example, some AI hardware accelerator architectures feature unique logic that is not ATPG-friendly, for example, a very deep logic depth at postsynthesis requesting very high coverage. Traditional test point insertion may lead to unacceptable performance or area penalty. To handle such issues, upgrades in ATPG algorithms or DFT architecture may be needed. Moreover, the extensive use of large-sized SRAMs at advanced tech-nodes may see new types of memory faults, and MBIST algorithms may need to be updated too.

In addition, as already a trend in traditional design, efficient and effective DFT activities need to be both left-shift and right-shift. Left-shift refers to the DFT involvement in early design stages, for example, DFT flow starts with design architecting and floor-planning, most DFT implementations are done at the RTL stage, and so on. Right-shift refers to the DFT activities extending well into or even beyond postsilicon stages such as bring up, diagnosis, volume production, and in-field debug, so that the entire product life-cycle quality can be ensured. Since most AI hardware accelerator designs are domain-specific with a very tight schedule, left-shift can help shrink the design cycle and meet the time-to-market target. Meanwhile, the new AI hardware accelerator architectures or components are usually not fully proved over their service time, thus a right-shift strategy is crucial to fully capture silicon characterization and product behavior over its life cycle. This learning is particularly important for mission-critical products such as automobiles and data centers, where reliability is of top concern and in-field tests may be dictated. From the DFT perspective, such designs not only require a complete solution from the regular scan, MBIST, I/O test, to online test, but also a close correlation between the test and function operation to ensure a high-quality product. This goal may be as challenging as the design itself.

Finally, as Moore's law slows down, 3-D IC has been proposed as a major solution to performance gain, cost reduction, and shrink of form factor. While traditional interposer-based 2.5-D solutions connect dies horizontally and have been widely adopted, 3-D designs stack dies in the vertical dimension and currently have focused mostly on external memories such as HBM. Tests for 3-D IC with memory dies have seen major advances in addressing several challenges. First, faults on through-silicon vias (TSV) and memories have to be tested. Second, these faults need diagnosis solutions for repair and quick yield ramp-up. Finally, an at-speed self-test is needed to reduce cost and ensure fault coverage. Moving forward, as stacked logic die becomes widely adopted, the test, diagnosis, and repair of interconnects between dies and test access of stacked dies seem to be the next challenge.

Dependability

From the perspective of AI hardware and system dependability, one fundamental challenge is on specifiability of AI-based functionality. The traditional dependability assurance, for example, safety assurance required by standards such as ISO 26262 or IEC 61508, is based on the assumption that there exists a full specification of the targeted functionality. These specifications are then used to guide risk analysis, dependability (e.g., safety) management, concept development, and validation activities. The full specification assumption holds valid for most traditional rule-based programmed approaches. On the other hand, AI-based functionality may not be fully specifiable. For example, the functionality of object recognition in autonomous driving applications can only partially be specified using rules. While the lack of full specification is exactly one basic driving factor of employing data-driven AI methods in these application domains, it creates a big challenge to dependability assurance, especially under the existing framework.

AI specifiability challenge is also related to its interpretability challenge, especially for DNNs. While there is a significant research effort on "Explainable AI," many advanced DNN models have hitherto remained noninterpretable. This characteristic of an AI-based system becomes an obstacle to directly applying traditional white-box verification and testing methods, which are common in traditional dependability assurance practices. As we have surveyed, most works in AI hardware dependability focus on fault tolerance techniques. While fault tolerance is an important means to mitigate dependability issues, verification and testing are vital components to meet the assurance requirements, especially from a standard compliance perspective (e.g., to meet ASIL requirements defined in ISO 26262). Existing FuSa standards, for example, ISO26262 and ISO21448, do not explicitly address the specific characteristics of the AI system. Lack of both specifiability and interpretability renders the challenges of applying a traditional ISO26262 style approach to AI systems, unless these obstacles are removed, or a new alternative approach is taken.

One outstanding challenge is how to define efficiently measurable metrics for the evaluation of the dependability of AI hardware and systems, both online and offline. Most of the recent work we have surveyed are based on DNN models, and the metrics used for dependability evaluation are primarily based on prediction accuracy. While this metric is suitable for offline analysis with ground-truth info available (e.g., fault criticality analysis), it is challenging to use the accuracy metric during online dependability management where real-time assessment is needed and often ground truth may not exist. For these types of applications, an alternative metric may be needed to quickly assess the dependability state of AI hardware and system.

Also, currently, most research has been focused on supervised learning for DNNs. There are other paradigms of AI/ML, including unsupervised learning and reinforcement learning. The dependability of AI hardware for these paradigms is still underexplored. Moreover, there is an emerging end-to-end DNN approach (e.g., DNN is trained to infer the control directly from sensor data inputs) in many AI application domains. How to define the appropriate metrics for such an approach remains a challenge.

Fault prevention and fault tolerance are current focus areas of the AI hardware dependability research field, as witnessed by this survey. Fault prediction may start attracting more attention in some dependability-critical applications, where proactive management is much desired. A growing interest is calling for more research in this direction. For example, applying an AI approach to fault prediction of AI hardware may be a good example of creating a virtuous cycle of "AI for AI."

Finally, from an AI hardware perspective, some AI hardware architectures are highly specified for training, while others target optimization for inference workload. While most recent dependability studies focus on inference AI hardware, the dependability assurance of AI hardware used for training also deserves attention, especially with the edge computing and federated learning paradigms where training is moved from cloud to edge devices.

In this article, we presented a systematic survey on state-of-the-art research and development of AI hardware testability and dependability. With the emergence of more hardware innovations to address AI computing challenges, testability and dependability challenges of AI hardware should be addressed to meet both manufacturing quality and in-field service assurance requirements. This article covers the research of this new field, which has rapidly been evolving especially over the past few years. Although much work has been done, in the future many open challenges remain, including fault criticality assessment with dramatically exploding fault space, hardware-aware fault modeling at the high abstraction level, practical DFT for large-scale designs and 3-D/2.5-D advanced packaging, dependability verification and validation with the limited specifiability and interpretability of AI models, efficient metrics for in-field real-time dependability evaluation, and so on. To address these existing and upcoming challenges, continuous advances of innovations from both industry and academia are expected over the next years. ■

Acknowledgments

The work of Haralampos-G. Stratigopoulos was supported by the ANR RE-TRUSTING Project under Grant ANR-21-CE24-0015-03.

References

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[2] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *ACM SIGARCH Comput. Archit. News*, vol. 23, no. 1, pp. 20–24, Mar. 1995.

[3] V. Sze et al., "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[4] A. Reuther et al., "Survey and benchmarking of machine learning accelerators," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Sep. 2019, pp. 1–9.

[5] M. Bouvier et al., "Spiking neural networks hardware implementations and challenges: A survey," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 2, pp. 1–35, Apr. 2019.

[6] Y. Chen et al., "A survey of accelerator architectures for deep neural networks," *Engineering*, vol. 6, no. 3, pp. 264–274, Mar. 2020.

[7] B. L. Deng et al., "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proc. IEEE*, vol. 108, no. 4, pp. 485–532, Apr. 2020.

[8] M. Capra et al., "An updated survey of efficient hardware architectures for accelerating deep convolutional neural networks," *Future Internet*, vol. 12, no. 7, p. 113, Jul. 2020.

[9] S. Bavikadi et al., "A survey on machine learning accelerators and evolutionary hardware platforms," *IEEE Design Test*, vol. 39, no. 3, pp. 91–116, Jun. 2022.

[10] M. Shafique et al., "An overview of next-generation architectures for machine learning: Roadmap, opportunities and challenges in the IoT era," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Mar. 2018, pp. 827–832.

[11] M. Costandi, *Neuroplasticity*. Cambridge, MA, USA: MIT Press, 2016.

[12] S. Dave et al., "Special session: Towards an agile design methodology for efficient, reliable, and secure ML systems," in *Proc. IEEE 40th VLSI Test Symp. (VTS)*, Apr. 2022, pp. 1–14.

[13] Y. LeCun et al., "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[15] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Netw.*, vol. 10, no. 9, pp. 1659–1671, Dec. 1997.

[16] C. Mead, *Analog VLSI and Neural Systems*. Reading, MA, USA: Addison-Wesley, 1989.

[17] G. Indiveri et al., "Neuromorphic silicon neuron circuits," *Frontiers Neurosci.*, vol. 5, May 2011, Art. no. 73.

[18] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: Opportunities and challenges," *Frontiers Neurosci.*, vol. 12, Oct. 2018, Art. no. 774.

[19] L. Camuñas-Mesa, B. Linares-Barranco, and T. Serrano-Gotarredona, "Neuromorphic spiking neural networks and their memristor-CMOS hardware implementations," *Materials*, vol. 12, no. 17, Aug. 2019, Art. no. 2745.

[20] M. Valle, "Analog VLSI implementation of artificial neural networks with supervised on-chip learning," *Analog Integr. Circuits Signal Process.*, vol. 33, pp. 263–287, Dec. 2002.

[21] S.-C. Liu et al., *Event-Based Neuromorphic Systems*. Hoboken, NJ, USA: Wiley, 2014.

[22] B. Chatterjee et al., "Exploiting inherent error resiliency of deep neural networks to achieve extreme energy efficiency through mixed-signal neurons," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 6, pp. 1365–1377, Jun. 2019.

[23] A. Rubino et al., "Ultra-low-power FDSOI neural circuits for extreme-edge neuromorphic intelligence," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 1, pp. 45–56, Jan. 2021.

[24] Z. Du et al., "Leveraging the error resilience of neural networks for designing highly energy efficient accelerators," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 8, pp. 1223–1235, Aug. 2015.

[25] S. Gupta et al., "Deep learning with limited numerical precision," in *Proc. Int. Conf. Mach. Learn. (ICML)*, Jul. 2015, pp. 1737–1746.

[26] M. Courbariaux et al., "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," Mar. 2016, *arXiv:1602.02830*.

[27] M. Rastegari et al., "XNOR-Net: ImageNet classification using binary convolutional neural networks," Aug. 2016, *arXiv:1603.05279*.

[28] T. P. Xiao et al., "Analog architectures for neural network acceleration based on non-volatile memory," *Appl. Phys. Rev.*, vol. 7, no. 3, Sep. 2020, Art. no. 031301.

[29] A. Ankit et al., "Circuits and architectures for in-memory computing-based machine learning accelerators," *IEEE Micro*, vol. 40, no. 6, pp. 8–22, Nov./Dec. 2020.

[30] S. Yu, "Neuro-inspired computing with emerging nonvolatile memories," *Proc. IEEE*, vol. 106, no. 2, pp. 260–285, Feb. 2018.

[31] B. Belhadj et al., "The improbable but highly appropriate marriage of 3D stacking and neuromorphic accelerators," in *Proc. Int. Conf. Compil., Archit. Synth. Embedded Syst. (CASES)*, Oct. 2014, pp. 1–9.

[32] D. Kim et al., "Neurocube: A programmable digital neuromorphic architecture with high-density 3D memory," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 380–392.

[33] S. Moradi et al., "A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs)," *IEEE Trans. Biomed. Circuits Syst.*, vol. 12, no. 1, pp. 106–122, Feb. 2018.

[34] J. Schemmel et al., "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *Proc. IEEE Int. Symp. Circuits Syst.*, May/Jun. 2010, pp. 1947–1950.

[35] B. V. Benjamin et al., "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proc. IEEE*, vol. 102, no. 5, pp. 699–716, May 2014.

[36] A. Valentian et al., "Fully integrated spiking neural network with analog neurons and RRAM synapses," in *IEDM Tech. Dig.*, Dec. 2019, pp. 14.3.1–14.3.4.

[37] G. K. Chen et al., "A 4096-neuron 1M-synapse 3.8-pJ/SOP spiking neural network with on-chip STDP learning and sparse weights in 10-nm FinFET CMOS," *IEEE J. Solid-State Circuits*, vol. 54, no. 4, pp. 992–1002, Apr. 2019.

[38] C. Frenkel et al., "A 0.086-mm^2 12.7-pJ/SOP 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm CMOS," *IEEE Trans. Biomed. Circuits Syst.*, vol. 13, no. 1, pp. 145–158, Feb. 2019.

[39] S. B. Furber et al., "The SpiNNaker project," *Proc. IEEE*, vol. 102, no. 5, pp. 652–665, May 2014.

[40] P. A. Merolla et al., "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, Aug. 2014.

[41] M. Davies et al., "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan./Feb. 2018.

[42] H. Mostafa et al., "Fast classification using sparsely active spiking networks," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017, pp. 1–4.

[43] L. A. Camuñas-Mesa et al., "A configurable event-driven convolutional node with rate saturation mechanism for modular ConvNet systems implementation," *Frontiers Neurosci.*, vol. 12, Feb. 2018, Art. no. 63.

[44] D. Bankman et al., "An always-on 3.8 µJ/86% CIFAR-10 mixed-signal binary CNN processor with all memory on chip in 28-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 54, no. 1, pp. 158–172, Jan. 2019.

[45] J. Zhang, Z. Wang, and N. Verma, "In-memory computation of a machine-learning classifier in a standard 6T SRAM array," *IEEE J. Solid-State Circuits*, vol. 52, no. 4, pp. 915–924, Apr. 2017.

[46] C. Eckert et al., "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 383–396.

[47] P. Chi et al., "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 27–39.

[48] A. Shafiee et al., "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 14–26.

[49] L. Song et al., "PipeLayer: A pipelined ReRAM-based accelerator for deep learning," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2017, pp. 541–552.

[50] C.-X. Xue et al., "A CMOS-integrated compute-in-memory macro based on resistive random-access memory for AI edge devices," *Nature Electron.*, vol. 4, no. 1, pp. 81–90, Dec. 2020.

[51] N. Nassif et al., "Sapphire rapids: The next-generation Intel Xeon scalable processor," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2022, pp. 44–46.

[52] Y. Chen et al., "Diannao family: Energy-efficient hardware accelerators for machine learning," *Commun. ACM*, vol. 59, no. 11, pp. 105–112, Nov. 2016.

[53] Y. Chen et al., "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[54] N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 1–12.

[55] C. Farabet et al., "CNP: An FPGA-based processor for convolutional networks," in *Proc. Int. Conf. Field Program. Log. Appl.*, Aug./Sep. 2009, pp. 32–37.

[56] Y. Umuroglu et al., "FINN: A framework for fast, scalable binarized neural network inference," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2017, pp. 65–74.

[57] G. Li et al., "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2017, pp. 1–12.

[58] E. M. E. Mhamdi and R. Guerraoui, “When neurons fail,” in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May/Jun. 2017, pp. 1028–1037.

[59] F. H. Bahnsen, V. Klebe, and G. Fey, “Effect analysis of low-level hardware faults on neural networks using emulated inference,” in *Proc. 10th Int. Conf. Modern Circuits Syst. Technol. (MOCAST)*, Jul. 2021, pp. 1–6.

[60] D. Maliuk et al., “Analog neural network design for RF built-in self-test,” in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2010, pp. 1–10, Paper 23.2.

[61] C. H. Sequin and R. D. Clay, “Fault tolerance in artificial neural networks,” in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, vol. 1, Jun. 1990, pp. 703–708.

[62] G. Bolt, “Fault models for artificial neural networks,” in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, vol. 2, Nov. 1991, pp. 1373–1378.

[63] P. Chandra and Y. Singh, “Fault tolerance of feedforward artificial neural networks—A framework of study,” in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, vol. 1, 2003, pp. 489–494.

[64] D. B. I. Feltham and W. Maly, “Physically realistic fault models for analog CMOS neural networks,” *IEEE J. Solid-State Circuits*, vol. 26, no. 9, pp. 1223–1229, Sep. 1991.

[65] A. S. Orgenci, G. Dundar, and S. Balkur, “Fault-tolerant training of neural networks in the presence of MOS transistor mismatches,” *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 48, no. 3, pp. 272–281, Mar. 2001.

[66] B. Reagen et al., “Ares: A framework for quantifying the resilience of deep neural networks,” in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018, pp. 1–6.

[67] Y. He, P. Balaprakash, and Y. Li, “FIdelity: Efficient resilience analysis framework for deep learning accelerators,” in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2020, pp. 270–281.

[68] Z. Chen et al., “BinFI: An efficient fault injector for safety-critical machine learning systems,” in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2019, pp. 1–23.

[69] L. M. Luza et al., “Emulating the effects of radiation-induced soft-errors for the reliability assessment of neural networks,” *IEEE Trans. Emerg. Topics Comput.*, vol. 10, no. 4, pp. 1867–1882, Oct./Dec. 2022.

[70] A. P. Arechiga and A. J. Michaels, “The robustness of modern deep learning architectures against single event upset errors,” in *Proc. High Perform. Extreme Comput. Conf. (HPEC)*, Sep. 2018, pp. 1–6.

[71] M. Sabbagh et al., “Evaluating fault resiliency of compressed deep neural networks,” in *Proc. IEEE Int. Conf. Embedded Softw. Syst. (ICESS)*, Jun. 2019, pp. 1–7.

[72] A. Bosio et al., “A reliability analysis of a deep neural network,” in *Proc. IEEE Latin Amer. Test Symp. (LATS)*, Mar. 2019, pp. 1–6.

[73] A. Ruospo et al., “Evaluating convolutional neural networks reliability depending on their data representation,” in *Proc. 23rd Euromicro Conf. Digit. Syst. Design (DSD)*, Aug. 2020, pp. 672–679.

[74] M. A. Neggaz et al., “Are CNNs reliable enough for critical applications? An exploratory study,” *IEEE Design Test*, vol. 37, no. 2, pp. 76–83, Apr. 2020.

[75] K. Givaki et al., “On the resilience of deep learning for reduced-voltage FPGAs,” in *Proc. 28th Euromicro Int. Conf. Parallel, Distrib. Netw.-Based Process. (PDP)*, Mar. 2020, pp. 110–117.

[76] Z. Gao et al., “Reliability evaluation of pruned neural networks against errors on parameters,” in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Oct. 2020, pp. 1–6.

[77] Y. Ibrahim et al., “Soft error resilience of deep residual networks for object recognition,” *IEEE Access*, vol. 8, pp. 19490–19503, 2020.

[78] K. Adam, I. I. Mohd, and Y. M. Younis, “The impact of the soft errors in convolutional neural network on GPUs: AlexNet as case study,” *Proc. Comput. Sci.*, vol. 182, pp. 89–94, Jan. 2021.

[79] E. Malekzadeh et al., “The impact of faults on DNNs: A case study,” in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Oct. 2021, pp. 1–6.

[80] Z. Wan et al., “Analyzing and improving fault tolerance of learning-based navigation systems,” in *Proc. 58th ACM/IEEE Design Autom. Conf. (DAC)*, Dec. 2021, pp. 841–846.

[81] F. F. D. Santos et al., “Analyzing and increasing the reliability of convolutional neural networks on GPUs,” *IEEE Trans. Rel.*, vol. 68, no. 2, pp. 663–677, Jun. 2019.

[82] F. F. D. Santos et al., “Impact of reduced precision in the reliability of deep neural networks for object detection,” in *Proc. IEEE Eur. Test Symp. (ETS)*, May 2019, pp. 1–6.

[83] A. Lotfi et al., “Resiliency of automotive object detection networks on GPU architectures,” in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2019, pp. 1–9.

[84] F. F. D. Santos et al., “Characterizing a neutron-induced fault model for deep neural networks,” *IEEE Trans. Nucl. Sci.*, early access, Nov. 24, 2022, doi: 10.1109/TNS.2022.3224538.

[85] F. Libano et al., “Selective hardening for neural networks in FPGAs,” *IEEE Trans. Nucl. Sci.*, vol. 66, no. 1, pp. 216–222, Jan. 2019.

[86] A. Mahmoud et al., “PyTorchFI: A runtime perturbation tool for DNNs,” in *Proc. 50th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. Workshops (DSN-W)*, Jun./Jul. 2020, pp. 25–31.

[87] Z. Chen et al., “TensorFI: A flexible fault injection framework for tensorflow applications,” in *Proc. IEEE Int. Symp. Softw. Rel. Eng. (ISSRE)*, Oct. 2020, pp. 426–435.

[88] C. Bolchini et al., “Fast and accurate error simulation for CNNs against soft errors,” *IEEE Trans. Comput.*, early access, Jun. 17, 2022, doi: 10.1109/TC.2022.3184274.

[89] Y. Zhang et al., “Estimating vulnerability of all model parameters in DNN with a small number of fault injections,” in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Mar. 2022, pp. 60–63.

[90] C. D. Schuman et al., “Resilience and robustness of spiking neural networks for neuromorphic systems,” in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2020, pp. 1–10.

[91] T. Spyrou et al., “Neuron fault tolerance in spiking neural networks,” in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Feb. 2021, pp. 743–748.

[92] B. Salami, O. S. Unsal, and A. C. Kestelman, “On the resilience of RTL NN accelerators: Fault characterization and mitigation,” in *Proc. 30th Int. Symp. Comput. Archit. High Perform. Comput. (SBAC-PAD)*, Sep. 2018, pp. 322–329.

[93] A. Ruospo et al., “A pipelined multi-level fault injector for deep neural networks,” in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Oct. 2020, pp. 1–6.

[94] J. E. R. Condia et al., “Combining architectural simulation and software fault injection for a fast and accurate CNNs reliability evaluation on GPUs,” in *Proc. IEEE 39th VLSI Test Symp. (VTS)*, Apr. 2021, pp. 1–7.

[95] J. J. Zhang, K. Basu, and S. Garg, “Fault-tolerant systolic array based accelerators for deep neural network execution,” *IEEE Design Test*, vol. 36, no. 5, pp. 44–53, Oct. 2019.

[96] J. Deng et al., “Retraining-based timing error mitigation for hardware neural networks,” in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, 2015, pp. 593–596.

[97] A. Chaudhuri et al., “Fault-criticality assessment for AI accelerators using graph convolutional networks,” in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Feb. 2021, pp. 1596–1599.

[98] A. Chaudhuri et al., “Functional criticality analysis of structural faults in AI accelerators,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 12, pp. 5657–5670, Dec. 2022.

[99] O. Temam, “A defect-tolerant accelerator for emerging high-performance applications,” in *Proc. 39th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2012, pp. 356–367.

[100] S. A. El-Sayed et al., “Spiking neuron hardware-level fault modeling,” in *Proc. IEEE 26th Int. Symp. On-Line Test. Robust Syst. Design (IOLTS)*, Jul. 2020, pp. 1–4.

[101] G. Gambardella et al., “Efficient error-tolerant quantized neural network accelerators,” in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Oct. 2019, pp. 1–6.

[102] C. De Sio, S. Azimi, and L. Sterpone, “An emulation platform for evaluating the reliability of deep neural networks,” in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Oct. 2020, pp. 1–4.

[103] N. Khoshavi, C. Broyles, and Y. Bi, “Compression or corruption? A study on the effects of transient faults on BNN inference accelerators,” in *Proc. 21st Int. Symp. Quality Electron. Design (ISQED)*, Mar. 2020, pp. 99–104.

[104] N. Khoshavi et al., “SHIELDeNN: Online accelerated framework for fault-tolerant deep neural network architectures,” in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2020, pp. 1–6.

[105] I. Souvatzoglou et al., “Analyzing the single event upset vulnerability of binarized neural networks on SRAM FPGAs,” in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Oct. 2021, pp. 1–6.

[106] D. Xu et al., “Reliability evaluation and analysis of FPGA-based neural network acceleration system,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 3, pp. 472–484, Mar. 2021.

[107] P. Corneliou et al., “Fine-grained vulnerability analysis of resource constrained neural inference accelerators,” in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Oct. 2021, pp. 1–6.

[108] T. Spyrou et al., “Reliability analysis of a spiking neural network hardware accelerator,” in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Mar. 2022, pp. 370–375.

[109] G. Abich et al., "Applying lightweight soft error mitigation techniques to embedded mixed precision deep neural networks," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 11, pp. 4772–4782, Nov. 2021.

[110] G. Abich et al., "The impact of soft errors in memory units of edge devices executing convolutional neural networks," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 3, pp. 679–683, Mar. 2022.

[111] L. M. Luza et al., "Investigating the impact of radiation-induced soft errors on the reliability of approximate computing systems," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Oct. 2020, pp. 1–6.

[112] F. Libano et al., "How reduced data precision and degree of parallelism impact the reliability of convolutional neural networks on FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 68, no. 5, pp. 865–872, May 2021.

[113] G. Gambardella et al., "Accelerated radiation test on quantized neural networks trained with fault aware training," in *Proc. IEEE Aerosp. Conf. (AERO)*, Mar. 2022, pp. 1–7.

[114] R. L. R. Junior et al., "High energy and thermal neutron sensitivity of Google tensor processing units," *IEEE Trans. Nucl. Sci.*, vol. 69, no. 3, pp. 567–575, Mar. 2022.

[115] C. Liu et al., "Rescuing memristor-based neuromorphic design with high defects," in *Proc. 54th Annu. Design Autom. Conf.*, Jun. 2017, pp. 1–6.

[116] E.-I. Vatajelu, G. Di Natale, and L. Anghel, "Special session: Reliability of hardware-implemented spiking neural networks (SNN)," in *Proc. IEEE 37th VLSI Test Symp. (VTS)*, Apr. 2019, pp. 1–8.

[117] Z. Ye et al., "Evaluation of radiation effects in RRAM-based neuromorphic computing system for inference," *IEEE Trans. Nucl. Sci.*, vol. 66, no. 1, pp. 97–103, Jan. 2019.

[118] C.-Y. Chen and K. Chakrabarty, "Efficient identification of critical faults in memristor crossbars for deep neural networks," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Feb. 2021, pp. 1074–1077.

[119] F. Chollet et al. (2015). *Keras*. [Online]. Available: https://keras.io

[120] J. Bergstra et al., "Theano: A CPU and GPU math compiler in python," in Proc. 9th Python Sci. Conf. (SciPy), Jan. 2010, pp. 18–24.

[121] M. Abadi et al., "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, Nov. 2016, pp. 265–283.

[122] S. K. S. Hari et al., "SASSIFI: An architecture-level fault injection tool for GPU application resilience evaluation," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Apr. 2017, pp. 249–258.

[123] T. Tsai et al., "NVBitFI: Dynamic fault injection for GPUs," in *Proc. 51st Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2021, pp. 284–291.

[124] D. Oliveira et al., "Experimental and analytical study of Xeon Phi reliability," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal. (SC)*, Nov. 2017, pp. 1–12.

[125] S. B. Shrestha and G. Orchard, "SLAYER: Spike layer error reassignment in time," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2018, pp. 1412–1421.

[126] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, vol. 32, H. Wallach et al., Eds. Red Hook, NY, USA: Curran Associates, 2019, pp. 8024–8035.

[127] S. Kundu et al., "Toward functional safety of systolic array-based deep learning hardware accelerators," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 3, pp. 485–498, Mar. 2021.

[128] S. Knowles, "Designing the colossus MK2 IPU," in *Proc. Hot Chips*, 2021.

[129] T. Norrie et al., "The design process for Google's training chips: TPUv2 and TPUv3," *IEEE Micro*, vol. 41, no. 2, pp. 56–63, Mar./Apr. 2021.

[130] G. Lauterbach, "The path to successful wafer-scale integration: The cerebras story," *IEEE Micro*, vol. 41, no. 6, pp. 52–57, Nov. 2021.

[131] G. Giles et al., "Test access mechanism for multiple identical cores," in *Proc. IEEE Int. Test Conf. (ITC)*, Oct. 2008, pp. 1–10.

[132] K. Chakravadhanula et al., "SmartScan—Hierarchical test compression for pin-limited low power designs," in *Proc. IEEE Int. Test Conf. (ITC)*, Sep. 2013, pp. 1–9.

[133] M. Sharma et al., "A novel test access mechanism for failure diagnosis of multiple isolated identical cores," in *Proc. IEEE Int. Test Conf.*, Sep. 2011, pp. 1–9.

[134] A. Chaudhuri et al., "C-testing and efficient fault localization for AI accelerators," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 7, pp. 2348–2361, Jul. 2022.

[135] S. Motaman, S. Ghosh, and J. Park, "A perspective on test methodologies for supervised machine learning accelerators," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 3, pp. 562–569, Sep. 2019.

[136] Y. Huang and R. Singhal, "Tutorial 1B: AI chip technologies and DFT methodologies," in *Proc. 32nd IEEE Int. Syst.-on-Chip Conf. (SOCC)*, Sep. 2019, pp. 1–2.

[137] H. Jia et al., "A programmable neural-network inference accelerator based on scalable in-memory computing," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2021, pp. 236–237.

[138] D. Niu et al., "184 QPS/W 64 Mb/mm^2 3D logic-to-DRAM hybrid bonding with process-near-memory engine for recommendation system," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2022, pp. 1–3.

[139] J.-F. Cote et al., "Streaming scan network (SSN): An efficient packetized data network for testing of complex SoCs," in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2020, pp. 1–10.

[140] G. Boschi et al., "Die-to-die testing and ECC error mitigation in automotive and industrial safety applications," in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2020, pp. 1–6.

[141] M. Hutner, G. Tshagharyan, and G. Harutyunyan, "Innovative practices on in-system test and reliability of memories," in *Proc. IEEE VLSI Test Symp. (VTS)*, Apr. 2019, p. 1.

[142] G. Harutyunyan and Y. Zorian, "An effective embedded test & diagnosis solution for external memories," in *Proc. IEEE 21st Int. On-Line Test. Symp. (IOLTS)*, Jul. 2015, pp. 168–170.

[143] S. Bandyopadhyay et al., "Innovative practices on in-system test and reliability of memories," in *Proc. IEEE 37th VLSI Test Symp. (VTS)*, Apr. 2019, p. 1.

[144] *IEEE Standard for Test Access Architecture for Three-Dimensional Stacked Integrated Circuits*, IEEE Standard 1838-2019, 2020, pp. 1–73.

[145] A. Gebregiorgis and M. B. Tahoori, "Testing of neuromorphic circuits: Structural vs functional," in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2019, pp. 1–10, Paper 3.2.

[146] W. Li et al., "RRAMedy: Protecting ReRAM-based neural network from permanent and soft faults during its lifetime," in *Proc. IEEE 37th Int. Conf. Comput. Design (ICCD)*, Nov. 2019, pp. 91–99.

[147] C.-Y. Chen and K. Chakrabarty, "On-line functional testing of memristor-mapped deep neural networks using backdoored checksums," in *Proc. IEEE Int. Test Conf. (ITC)*, Oct. 2021, pp. 83–92.

[148] O. Aramoon and G. Qu, "Provably accurate memory fault detection method for deep neural networks," in *Proc. Great Lakes Symp. VLSI (GLSVLSI)*, Jun. 2021, pp. 443–448.

[149] S. T. Ahmed and M. B. Tahoori, "Compact functional test generation for memristive deep learning implementations using approximate gradient ranking," in *Proc. IEEE Int. Test Conf. (ITC)*, Sep. 2022, pp. 239–248.

[150] H.-Y. Tseng et al., "Machine learning-based test pattern generation for neuromorphic chips," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Nov. 2021, pp. 1–7.

[151] S. A. El-Sayed et al., "Compact functional testing for neuromorphic computing circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, early access, Nov. 21, 2022, doi: 10.1109/TCAD.2022.3223843.

[152] B. Luo et al., "On functional test generation for deep neural network IPs," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Mar. 2019, pp. 1010–1015.

[153] K. Pei et al., "DeepXplore: Automated whitebox testing of deep learning systems," in *Proc. 26th Symp. Operating Syst. Princ. (SOSP)*, Oct. 2017, pp. 1–18.

[154] Y. Tian et al., "DeepTest: Automated testing of deep-neural-network-driven autonomous cars," in *Proc. 40th Int. Conf. Softw. Eng.*, May/Jun. 2018, pp. 303–314.

[155] K. Ma et al., "Efficient low cost alternative testing of analog crossbar arrays for deep neural networks," in *Proc. IEEE Int. Test Conf. (ITC)*, Sep. 2022, pp. 499–503.

[156] S. A. El-Sayed et al., "Self-testing analog spiking neuron circuit," in *Proc. 16th Int. Conf. Synth., Modeling, Anal. Simulation Methods Appl. Circuit Design (SMACD)*, Jul. 2019, pp. 81–84.

[157] A. Pandey et al., "Novel technique for manufacturing & in-system testing of large scale SoC using functional protocol based high-speed I/O," in *Proc. IEEE 40th VLSI Test Symp. (VTS)*, Apr. 2022, pp. 1–7.

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

[159] T. Uezono, Y. He, and Y. Li, "Achieving automotive safety requirements through functional in-field self-test for deep learning accelerators," in *Proc. IEEE Int. Test Conf. (ITC)*, Sep. 2022, pp. 465–473.

[160] E. Ozen and A. Orailoglu, "Low-cost error detection in deep neural network accelerators with linear algorithmic checksums," *J. Electron. Test., Theory Appl.*, vol. 36, no. 6, pp. 703–718, Dec. 2020.

[161] E. Ozen and A. Orailoglu, “Concurrent monitoring of operational health in neural networks through balanced output partitions,” in *Proc. 25th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2020, pp. 169–174.

[162] S. K. S. Hari et al., “Making convolutions resilient via algorithm-based error detection techniques,” *IEEE Trans. Depend. Sec. Comput.*, vol. 19, no. 4, pp. 2546–2558, Jul./Aug. 2022.

[163] B. F. Goldstein et al., “A lightweight error-resiliency mechanism for deep neural networks,” in *Proc. 22nd Int. Symp. Quality Electron. Design (ISQED)*, Apr. 2021, pp. 311–316.

[164] N. I. Deligiannis et al., “Towards the integration of reliability and security mechanisms to enhance the fault resilience of neural networks,” *IEEE Access*, vol. 9, pp. 155998–156012, 2021.

[165] A. Ruospo et al., “A suitability analysis of software based testing strategies for the on-line testing of artificial neural networks applications in embedded devices,” in *Proc. IEEE 27th Int. Symp. On-Line Test. Robust Syst. Design (IOLTS)*, Jun. 2021, pp. 1–6.

[166] M. Liu and K. Chakrabarty, “Online fault detection in ReRAM-based computing systems by monitoring dynamic power consumption,” in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2020, pp. 1–10.

[167] M. Sadi and U. Guin, “Test and yield loss reduction of AI and deep learning accelerators,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 7, pp. 1124–1135, Jan. 2021.

[168] L. Xia et al., “Stuck-at fault tolerance in RRAM computing systems,” *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 8, no. 1, pp. 102–115, Mar. 2018.

[169] Z. Song et al., “ITT-RNA: Imperfection tolerable training for RRAM-crossbar-based deep neural-network accelerator,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 1, pp. 129–142, Jan. 2021.

[170] L.-H. Hoang, M. A. Hanif, and M. Shafique, “TRe-map: Towards reducing the overheads of fault-aware retraining of deep neural networks by merging fault maps,” in *Proc. 24th Euromicro Conf. Digit. Syst. Design (DSD)*, Sep. 2021, pp. 434–441.

[171] S. Kannan et al., “Sneak-path testing of crossbar-based nonvolatile random access memories,” *IEEE Trans. Nanotechnol.*, vol. 12, no. 3, pp. 413–426, May 2013.

[172] C.-Y. Chen et al., “RRAM defect modeling and failure analysis based on march test and a novel squeeze-search scheme,” *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 180–190, Jan. 2015.

[173] E. I. Vatajelu et al., “Challenges and solutions in emerging memory testing,” *IEEE Trans. Emerg. Topics Comput.*, vol. 7, no. 3, pp. 493–506, Jul./Sep. 2019.

[174] L. Wu et al., “Defect and fault modeling framework for STT-MRAM testing,” *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 2, pp. 707–723, Apr./Jun. 2021.

[175] P. Liu et al., “Fault modeling and efficient testing of memristor-based memory,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 11, pp. 4444–4455, Nov. 2021.

[176] P. Girard et al., “A survey of test and reliability solutions for magnetic random access memories,” *Proc. IEEE*, vol. 109, no. 2, pp. 149–169, Feb. 2021.

[177] L. Wu et al., “Characterization, modeling, and test of intermediate state defects in STT-MRAMs,” *IEEE Trans. Comput.*, vol. 71, no. 9, pp. 2219–2233, Sep. 2022.

[178] M. Fieback et al., “Defects, fault modeling, and test development framework for RRAMs,” *ACM J. Emerg. Technol. Comput. Syst.*, vol. 18, no. 3, pp. 1–26, Apr. 2022.

[179] T. Han, I. Choi, and S. Kang, “Majority-based test access mechanism for parallel testing of multiple identical cores,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 8, pp. 1439–1447, Aug. 2015.

[180] A. Ramdas and O. Sinanoglu, “Testing chips with spare identical cores,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 7, pp. 1124–1135, Jul. 2013.

[181] P. N. Variyam, S. Cherubal, and A. Chatterjee, “Prediction of analog performance parameters using fast transient testing,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 21, no. 3, pp. 349-361, Mar. 2002.

[182] H.-G. Stratigopoulos and S. Mir, “Adaptive alternate analog test,” *IEEE Design Test Comput.*, vol. 29, no. 4, pp. 71–79, Aug. 2012.

[183] A. Avizienis et al., “Basic concepts and taxonomy of dependable and secure computing,” *IEEE Trans. Depend. Sec. Comput.*, vol. 1, no. 1, pp. 11–33, Jan. 2004.

[184] G. Buja and R. Menis, “Dependability and functional safety: Applications in industrial electronics systems,” *IEEE Ind. Electron. Mag.*, vol. 6, no. 3, pp. 4–12, Sep. 2012.

[185] *Road Vehicles-Functional Safety,* Standard ISO 26262, 2018.

[186] *Edition 2.0 Functional Safety*, Standard IEC 61508, 2010.

[187] *Software Considerations in Airborne Systems and Equipment Certification*, Standard RTCA/DO-178C, 2012.

[188] *Road Vehicles—Safety of the Intended Functionality*, Standard ISO PAS 21448, 2019.

[189] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture," *Commun. ACM*, vol. 62, no. 2, pp. 48–60, Feb. 2019.

[190] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," Mar. 2015, *arXiv:1412.6572*.

[191] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," *IEEE Access*, vol. 6, pp. 14410–14430, 2018.

[192] T. Graepel, K. Lauter, and M. Naehrig, "ML confidential: Machine learning on encrypted data," in *Proc. Int. Conf. Inf. Secur. Cryptol. (ICISC)*, Dec. 2012, pp. 1–21.

[193] B. Wang and N. Z. Gong, "Stealing hyperparameters in machine learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2018, pp. 36–52.

[194] W. Hua, Z. Zhang, and G. E. Suh, "Reverse engineering convolutional neural networks through side-channel information leaks," in *Proc. 55th Annu. Design Autom. Conf. (DAC)*, Jun. 2018, pp. 1–6.

[195] T. Gu et al., "BadNets: Evaluating backdooring attacks on deep neural networks," *IEEE Access*, vol. 7, pp. 47230–47244, 2019.

[196] H. Chen et al., "DeepInspect: A black-box trojan detection and mitigation framework for deep neural networks," in *Proc. 28th Int. Joint Conf. Artif. Intell. (IJCAI)*, Jul. 2019, pp. 4658–4664.

[197] Y. Liu et al., "Fault injection attack on deep neural network," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2017, pp. 131–138.

[198] A. S. Rakin, Z. He, and D. Fan, "Bit-flip attack: Crushing neural network with progressive bit search," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct./Nov. 2019, pp. 1211–1220.

[199] E. Cheng et al., "Cross-layer resilience: Challenges, insights, and the road ahead," in Proc. 56th ACM/IEEE Design Autom. Conf. (DAC), Jun. 2019, pp. 1–4.

[200] S. Mittal and J. S. Vetter, "A survey of CPU-GPU heterogeneous computing techniques," *ACM Comput. Surv.*, vol. 47, no. 4, p. 135, Jul. 2015.

[201] European Commission. (Apr. 2019). *Ethics Guidelines for Trustworthy Artificial Intelligence (AI)*. [Online]. Available: https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai

[202] C.-T. Chin et al., "Training techniques to obtain fault-tolerant neural networks," in *Proc. IEEE 24th Int. Symp. Fault-Tolerant Comput.*, Jun. 1994, pp. 360–369.

[203] B. S. Arad and A. El-Amawy, "On fault tolerant training of feedforward neural networks," *Neural Netw.*, vol. 10, no. 3, pp. 539–553, Apr. 1997.

[204] N. Wei, S. Yang, and S. Tong, "A modified learning algorithm for improving the fault tolerance of BP networks," in *Proc. Int. Conf. Neural Netw. (ICNN)*, vol. 1, Jun. 1996, pp. 247–252.

[205] P. J. Edwards and A. F. Murray, "Penalty terms for fault tolerance," in *Proc. Int. Conf. Neural Netw. (ICNN)*, vol. 2, Jun. 1997, pp. 943–947.

[206] S. Cavalieri and O. Mirabella, "A novel learning algorithm which improves the partial fault tolerance of multilayer neural networks," *Neural Netw.*, vol. 12, no. 1, pp. 91–106, Jan. 1999.

[207] C. Neti, M. H. Schneider, and E. D. Young, "Maximally fault tolerant neural networks," *IEEE Trans. Neural Netw.*, vol. 3, no. 1, pp. 14–23, Jan. 1992.

[208] D. Deodhare, M. Vidyasagar, and S. S. Keethi, "Synthesis of fault-tolerant feedforward neural networks using minimax optimization," *IEEE Trans. Neural Netw.*, vol. 9, no. 5, pp. 891–900, Sep. 1998.

[209] Z.-H. Zhou and S.-F. Chen, "Evolving fault-tolerant neural networks," *Neural Comput. Appl.*, vol. 11, nos. 3–4, pp. 156–160, Jun. 2003.

[210] E. Sugawara, M. Fukushi, and S. Horiguchi, "Fault tolerant multi-layer neural networks with GA training," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Syst. (DFT)*, Nov. 2003, pp. 328–335.

[211] N. C. Hammadi et al., "Fault tolerant constructive algorithm for feedforward neural networks," in *Proc. IEEE Pacific Rim Int. Symp. Fault-Tolerant Syst. (PRFTS)*, Dec. 1997, pp. 215–220.

[212] C. Torres-Huitzil and B. Girau, "Fault and error tolerance in neural networks: A review," *IEEE Access*, vol. 5, pp. 17322–17341, 2017.

[213] R. A. Solovyev, A. L. Stempkovsky, and D. V. Telpukhov, "Study of fault tolerance methods for hardware implementations of convolutional neural networks," *Opt. Memory Neural Netw.*, vol. 28, no. 2, pp. 82–88, Apr. 2019.

[214] G. B. Hacene et al., "Training modern deep neural networks for memory-fault robustness," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2019, pp. 1–5.

[215] U. Zahid et al., "FAT: Training neural networks for reliable inference under hardware faults," in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2020, pp. 1–10.

[216] L. Yang and B. Murmann, “SRAM voltage scaling for energy-efficient convolutional neural networks,” in *Proc. 18th Int. Symp. Quality Electron. Design (ISQED)*, Mar. 2017, pp. 7–12.

[217] R. V. W. Putra, M. A. Hanif, and M. Shafique, “SparkXD: A framework for resilient and energy-efficient spiking neural network inference using approximate DRAM,” in *Proc. 58th ACM/IEEE Design Autom. Conf. (DAC)*, Dec. 2021, pp. 379–384.

[218] Z. He et al., “Noise injection adaption: End-to-end ReRAM crossbar non-ideal effect adaption for neural network mapping,” in *Proc. 56th ACM/IEEE Annu. Design Autom. Conf.*, Jun. 2019, pp. 1–6.

[219] Y. Long, X. She, and S. Mukhopadhyay, “Design of reliable DNN accelerator with un-reliable ReRAM,” in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Mar. 2019, pp. 1769–1774.

[220] V. Joshi et al., “Accurate deep neural network inference using computational phase-change memory,” *Nature Commun.*, vol. 11, no. 1, pp. 1–13, May 2020.

[221] Y. Zhu et al., “Statistical training for neuromorphic computing using memristor-based crossbars considering process variations and noise,” in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Mar. 2020, pp. 1590–1593.

[222] D. Gaol et al., “Reliable memristor-based neuromorphic design using variation- and defect-aware training,” in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Nov. 2021, pp. 1–9.

[223] E. Ozen and A. Orailoglu, “SNR: Squeezing numerical range defuses bit error vulnerability surface in deep neural networks,” *ACM Trans. Embedded Comput. Syst.*, vol. 20, no. 5, pp. 1–25, Sep. 2021.

[224] M. Bocquet et al., “Embracing the unreliability of memory devices for neuromorphic computing,” in *Proc. IEEE Int. Rel. Phys. Symp. (IRPS)*, Apr./May 2020, pp. 1–5.

[225] A. Azizimazreah et al., “Tolerating soft errors in deep learning accelerators with reliable on-chip memory designs,” in *Proc. IEEE Int. Conf. Netw., Archit. Storage (NAS)*, Oct. 2018, pp. 1–10.

[226] C. Khunasaraphan, K. Vanapipat, and C. Lursinsap, “Weight shifting techniques for self-recovery neural networks,” *IEEE Trans. Neural Netw.*, vol. 5, no. 4, pp. 651–658, Jul. 1994.

[227] E. Ozen and A. Orailoglu, “Just say zero: Containing critical bit-error propagation in deep neural networks with anomalous feature suppression,” in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, Nov. 2020, pp. 1–9.

[228] E. Ozen and A. Orailoglu, “Boosting bit-error resilience of DNN accelerators through median feature selection,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 11, pp. 3250–3262, Nov. 2020.

[229] M. D. Emmerson and R. I. Damper, “Determining and improving the fault tolerance of multilayer perceptrons in a pattern-recognition application,” *IEEE Trans. Neural Netw.*, vol. 4, no. 5, pp. 788–793, Sep. 1993.

[230] C.-T. Chiu et al., “Robustness of feedforward neural networks,” in *Proc. IEEE Int. Conf. Neural Netw. (ICNN)*, vol. 2, Mar./Apr. 1993, pp. 783–788.

[231] T. Liu et al., “A fault-tolerant neural network architecture,” in *Proc. 56th ACM/IEEE Design Autom. Conf. (DAC)*, Jun. 2019, pp. 1–6.

[232] M. A. Hanif and M. Shafique, “DNN-life: An energy-efficient aging mitigation framework for improving the lifetime of on-chip weight memories in deep neural network hardware architectures,” in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Feb. 2021, pp. 729–734.

[233] A. Hashmi et al., “Automatic abstraction and fault tolerance in cortical microachitectures,” in *Proc. 38th Annu. Int. Symp. Comput. Archit.*, Jun. 2011, pp. 1–10.

[234] W. Li et al., “FTT-NAS: Discovering fault-tolerant neural architecture,” in *Proc. 25th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2020, pp. 211–216.

[235] L.-H. Hoang, M. A. Hanif, and M. Shafique, “FT-ClipAct: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation,” in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Mar. 2020, pp. 1241–1246.

[236] Z. Chen, G. Li, and K. Pattabiraman, “A low-cost fault corrector for deep neural networks through range restriction,” in *Proc. 51st Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2021, pp. 1–13.

[237] B. Ghavami et al., “FitAct: Error resilient deep neural networks via fine-grained post-trainable activation functions,” in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Mar. 2022, pp. 1239–1244.

[238] S. Burel, A. Evans, and L. Anghel, “Improving DNN fault tolerance in semantic segmentation applications,” in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Oct. 2022, pp. 1–6.

[239] Z. Xu and J. Abraham, “Safety design of a convolutional neural network accelerator with error localization and correction,” in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2019, pp. 1–10, Paper 12.3.

[240] M. Liu et al., “Algorithmic fault detection for RRAM-based matrix operations,” *ACM Trans. Design Autom. Electron. Syst.*, vol. 25, no. 3, pp. 29:1–29:31, May 2020.

[241] K. Zhao et al., “FT-CNN: Algorithm-based fault tolerance for convolutional neural networks,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 7, pp. 1677–1689, Jul. 2021.

[242] J. Kosaian and K. V. Rashmi, “Arithmetic-intensity-guided fault tolerance for neural network inference on GPUs,” in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal. (SC)*, Nov. 2021, pp. 1–15.

[243] D. Filippas et al., “Low-cost online convolution checksum checker,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 30, no. 2, pp. 201–212, Feb. 2022.

[244] C. S. Mummidi et al., “A highly-efficient error detection technique for general matrix multiplication using tiled processing on SIMD architecture,” in *Proc. IEEE 40th Int. Conf. Comput. Design (ICCD)*, Oct. 2022, pp. 529–536.

[245] C. Schorn, A. Guntoro, and G. Ascheid, “Accurate neuron resilience prediction for a flexible reliability management in neural network accelerators,” in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Mar. 2018, pp. 979–984.

[246] M. A. Hanif and M. Shafique, “SalvageDNN: Salvaging deep neural network accelerators with permanent faults through saliency-driven fault-aware mapping,” *Philos. Trans. Roy. Soc. A, Math., Phys. Eng. Sci.*, vol. 378, no. 2164, Feb. 2020, Art. no. 20190164.

[247] A. Ruospo and E. Sanchez, “On the reliability assessment of artificial neural networks running on AI-oriented MPSoCs,” *Appl. Sci.*, vol. 11, no. 14, p. 6455, Jul. 2021.

[248] R. V. W. Putra, M. A. Hanif, and M. Shafique, “ReSpawn: Energy-efficient fault-tolerance for spiking neural networks considering unreliable memories,” in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Nov. 2021, pp. 1–9.

[249] S.-K. Lu et al., “Fault resilience techniques for flash memory of DNN accelerators,” in *Proc. IEEE Int. Test Conf. Asia (ITC-Asia)*, Sep. 2022, pp. 591–600.

[250] E. Talpes et al., “Compute solution for Tesla’s full self-driving computer,” *IEEE Micro*, vol. 40, no. 2, pp. 25–35, Mar. 2020.

[251] K. Matsubara et al., “A 12 nm autonomous-driving processor with 60.4 TOPS, 13.8 TOPS/W CNN executed by task-separated ASIL D control,” in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2021, pp. 56–58.

[252] M. Abdulrahman et al., “HarDNN: Feature map vulnerability evaluation in CNNs,” *CoRR*, vol. abs/2002.09786, pp. 1–14, Dec. 2020.

[253] H. R. Mahdiani, S. M. Fakhraie, and C. Lucas, “Relaxed fault-tolerant hardware implementation of neural networks in the presence of multiple transient errors,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 8, pp. 1215–1228, Aug. 2012.

[254] A. Ruospo et al., “Selective hardening of critical neurons in deep neural networks,” in *Proc. 25th Int. Symp. Design Diagnostics Electron. Circuits Syst. (DDECS)*, Apr. 2022, pp. 136–141.

[255] C. Liu et al., “HyCA: A hybrid computing architecture for fault-tolerant deep learning,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 10, pp. 3400–3413, Oct. 2022.

[256] Z. Gao et al., “Soft error tolerant convolutional neural networks on FPGAs with ensemble learning,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 30, no. 3, pp. 291–302, Mar. 2022.

[257] B. Reagen et al., “Minerva: Enabling low-power, highly-accurate deep neural network accelerators,” in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 267–278.

[258] W. Li et al., “Soft error mitigation for deep convolution neural network on FPGA accelerators,” in *Proc. 2nd IEEE Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Aug. 2020, pp. 1–5.

[259] S. Burel, A. Evans, and L. Anghel, “MOZART+: Masking outputs with zeros for improved architectural robustness and testing of DNN accelerators,” *IEEE Trans. Device Mater. Rel.*, vol. 22, no. 2, pp. 120–128, Jun. 2022.

[260] S. Burel, A. Evans, and L. Anghel, “Zero-overhead protection for CNN weights,” in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Oct. 2021, pp. 1–6.

[261] R. V. W. Putra, M. A. Hanif, and M. Shafique, “SoftSNN: Low-cost fault tolerance for spiking neural network accelerators under soft errors,” in *Proc. 59th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2022, pp. 151–156.

[262] A. Agrawal, C. Lee, and K. Roy, “X-CHANGR: changing memristive crossbar mapping for mitigating line-resistance induced accuracy degradation in deep

neural networks," *CoRR*, vol. abs/1907.00285, pp. 1–8, Jun. 2019.

[263] T. Titirsha et al., "Endurance-aware mapping of spiking neural networks to neuromorphic hardware," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 2, pp. 288–301, Feb. 2022.

[264] S.-S. Lee and J.-S. Yang, "Value-aware parity insertion ECC for fault-tolerant deep neural network," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Mar. 2022, pp. 724–729.

[265] C. Schorn, A. Guntoro, and G. Ascheid, "Efficient on-line error detection and mitigation for deep neural network accelerators," in *Proc. Int. Conf. Comput. Saf. Rel. Secur. (SAFECOMP)*, Sep. 2018, pp. 205–219.

[266] S. Kim et al., "MATIC: Learning around errors for efficient low-voltage neural network accelerators," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Mar. 2018, pp. 1–6.

[267] P. N. Whatmough et al., "A 28 nm SoC with a 1.2 GHz 568 nJ/prediction sparse deep-neural-network engine with >0.1 timing error rate tolerance for IoT applications," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 242–243.

[268] J. Zhang et al., "ThUnderVolt: Enabling aggressive voltage underscaling and timing error resilience for energy efficient deep learning accelerators," in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018, pp. 1–6.

[269] A. P. Johnson et al., "Homeostatic fault tolerance in spiking neural networks: A dynamic hardware perspective," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 2, pp. 687–699, Feb. 2018.

[270] L. Xia et al., "Fault-tolerant training enabled by on-line fault detection for RRAM-based neural computing systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 9, pp. 1611–1624, Sep. 2019.

[271] S. Zhang et al., "Aging-aware lifetime enhancement for memristor-based neuromorphic computing," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Mar. 2019, pp. 1751–1756.

[272] P. I. Vaz et al., "Improving TID radiation robustness of a CMOS OxRAM-based neuron circuit by using enclosed layout transistors," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 6, pp. 1122–1131, Jun. 2021.

[273] M. V. Beigi and G. Memik, "Thermal-aware optimizations of ReRAM-based neuromorphic computing systems," in *Proc. 55th Annu. Design Autom. Conf. (DAC)*, Jun. 2018, pp. 1–6.

[274] H. Shin, M. Kang, and L.-S. Kim, "A thermal-aware optimization framework for ReRAM-based deep neural network acceleration," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, Nov. 2020, pp. 1–9.

[275] J. Meng et al., "Temperature-resilient RRAM-based in-memory computing for DNN inference," *IEEE Micro*, vol. 42, no. 1, pp. 89–98, Jan./Feb. 2022.

[276] N. Srivastava et al., "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jun. 2014.

[277] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Apr. 2017, pp. 1–16.

[278] D. Ernst et al., "Razor: Circuit-level correction of timing errors for low-power operation," *IEEE Micro*, vol. 24, no. 6, pp. 10–20, Nov./Dec. 2004.

[279] K.-H. Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Trans. Comput.*, vol. C-33, no. 6, pp. 518–528, Jun. 1984.

Fei Su is a DFX and telemetry architect at Intel Corporation, Folsom, CA 95630 USA. His research interests include testability and dependability of semiconductor circuits/chiplets, AI/ML hardware, cyber–physical systems, and edge/cloud computing. Su has a PhD from Duke University, Durham, NC, USA. He is a Senior Member of IEEE.

Chunsheng Liu is the leader of the DFT Team at Alibaba Inc, Sunnyvale, CA 94085 USA. His research interests include test infrastructure for high-performance processors, FPGA, and machine-learning accelerators, as well as high dependability of cloud computing hardware. Liu has a PhD from Duke University, Durham, NC, USA. He is a Senior Member of IEEE.

Haralampos-G. Stratigopoulos is a research director of the French National Center for Scientific Research (CNRS) at the LIP6 Laboratory, Sorbonne Université, 75005 Paris, France. His research interests include neuromorphic computing, hardware security, and design-for-test of integrated circuits and systems. Stratigopoulos has a PhD from Yale University, New Haven, CT, USA. He is a Member of IEEE.

■ Direct questions and comments about this article to Haralampos-G. Stratigopoulos, Sorbonne Université, CNRS, LIP6, France; haralampos.stratigopoulos@lip6.fr.

Shaping Resilient AI Hardware Through DNN Computational Feature Exploitation

Elbruz Ozen and Alex Orailoglu
Department of Computer Science and Engineering
University of California at San Diego
La Jolla, CA 92093 USA

Editor's notes:
This article presents a method that allows deep neural networks (DNNs) to learn and construct self-checking mechanisms so as to detect and suppress large magnitude hardware errors.

—Fei Su, Intel Corporation

Deep learning has become an indispensable part of our modem infrastructure in the past decade. Our transportation, healthcare devices, and industrial control systems will heavily rely on deep learning algorithms in the near future, portending the immense potential for delivering solutions to the fundamental challenges of humankind. The safety and reliability of deep learning accelerators have been promoted to the ranks of essential design considerations as the adoption of deep learning techniques in safety-critical application domains, including autonomous driving, healthcare devices, robotics, and industrial control systems, grows apace. Exposure to harsh environmental conditions or design marginalities may lift the likelihood of hardware errors considerably; the error effects might result in disastrous consequences and even threaten human life. Dedicated functional safety mechanisms are consequently necessitated in these domains to thwart such catastrophic scenarios.

Error tolerance in deep neural networks (DNNs) is governed by rules that differ from those of general-purpose computing. It is widely acknowledged that neural networks can maintain accuracy, to a large extent, under small error perturbations even when such error effects widely diffuse throughout the model. Nevertheless, hardware errors with numerical impacts that stretch beyond the prescribed perturbation levels constitute the Achilles' heel for DNNs, as even a handful of large-magnitude errors can noticeably deteriorate classification accuracy [1]. The comprehension of the inherent resilience and vulnerability characteristics of DNNs is therefore essential for error resilience integration into deep learning hardware at palatable costs.

What makes this investigation for novel fault tolerance promising is the inherent resilience of neural networks to minor perturbations together with the learning flexibility of deep learning models even when constricted by imposed constraints. This flexibility of neural networks affords the construction of novel error identification mechanisms by shaping the inherent redundancy of deep learning

Digital Object Identifier 10.1109/MDAT.2022.3156016
Date of publication: 2 March 2022; date of current version: 10 March 2023.

algorithms. Moreover, the resilience of neural networks to minor perturbations opens up opportunities for approximate error mitigation without having to pay for perfect value restoration.

Innovative approaches for introspective error localization and approximate error amelioration are essential to reap the benefits of the outlined computational characteristics of neural networks. Instead of conventional and costly error detection techniques, DNNs can identify critical hardware bit errors via self-error-checking invariants learned through the training process. We demonstrate that error detection invariants can be crafted at various granularities based on the needs of the application to enable low-cost error detection and precise error localization in the deep learning hardware datapath while necessitating no additional information redundancy.

In lieu of precise correction of the error perturbations, the impact of bit errors can be largely contained through anomaly[1] suppression, which is carried out by dropping or filtering variables that are contaminated with errors. The extreme effectiveness of these methods in maintaining model accuracy is experimentally established even at high error rates, while the implementation of the aforementioned techniques rarely incurs perceptible costs nor necessitates any information redundancy to deliver such error rectification. By simply snapping outsized error effects back to within the realm of minor numerical inaccuracies, procedures such as dropping erroneous values squash the error impact effectively and improve the bit error tolerance of deep learning algorithms by exploiting their inherent resilience characteristics to limited magnitude perturbations and complementing their inherent sparsity.

The approximate and resilient nature of DNNs yields the possibility of large-scale and efficient accuracy preservation by prioritizing the large-magnitude bit errors and effectively attenuating them through the outlined approach. Such a perspective foreshadows fundamental breakthroughs for the error resilience problem in deep learning hardware and leads to strong functional safety characteristics at almost negligible costs.

Related work

Traditional fault tolerance methods are frequently utilized in safety-critical electronics. Parity bits and error correction codes (ECCs) can protect data integrity in memory elements, and modular redundancy techniques are frequently employed in the execution path. Traditional fault tolerance mechanisms, particularly modular redundancy techniques, incur significant area and power overheads that may prove prohibitive for the resource-constrained deep learning hardware platforms.

Fault tolerance methods specific to neural networks have recently attracted attention as they promise strong error resilience characteristics at much more palatable costs in deep learning hardware. Novel error detection techniques such as symptom-based detectors [2] rely on the expected distribution of neural network variables to identify anomalies. Algorithmic checksums [1] can detect bit errors in neural networks by utilizing the linearity property of convolutional and fully connected layers. Novel recent studies [3], [4] demonstrate that algorithmic consistency checks can be encoded in neural networks through the training process, and error detection can be achieved through additional checker neurons. The inherent resilience characteristics of neural networks can be boosted through various error mitigation techniques, including activation range restriction [5] to attenuate large magnitude errors, and training the deep learning models in the presence of faults [6] to adapt to error perturbations. Moreover, computational fabric particularities can be utilized to innovate the testing techniques for deep learning accelerators [7].

Overview of neural network characteristics

This section presents a brief tutorial on DNN computational characteristics that are of fundamental importance to the construction of novel error detection and mitigation techniques in deep learning hardware.

The behavior of DNNs is determined by the parameter configuration learned through the training process. The training procedure is carried out by modifying the model parameters at each step to minimize the loss function, which measures the closeness of the target value and the produced output by the neural network. Additional behavioral goals can thus be embedded into DNNs by augmenting the loss function with well-crafted supplementary terms for the desired goals. To illustrate, two parameters can be forced to be as numerically close as possible

[1]The term "anomaly" in this article refers to cases where a variable diverges from its learned and expected value due to a hardware error.

simply by adding their distance ($|w_1 - w_2|$) to the loss function as an additional term. Such techniques constitute a highly effective approach for embedding invariants into DNNs, thus enabling the error-caused violation of such invariants at inference time to serve as practical error detection mechanisms.

Moreover, the flexibility of the training process in neural networks spawns a diverse set of models of comparable accuracy levels, even including ones that fulfill strict constraints imposed on the computational graph. For instance, the forward pass of the model can be modified to rein in the propagated variable magnitude if the observed magnitude is unusually larger than the expected value. Neural networks can be trained effectively under graph constraints as long as the necessary information represented in the forward pass and the backward pass is able to be carried out efficiently. Invariants embedded into neural networks through the imposed graph constraints can then be utilized for error identification at inference time.

A few other essential properties of DNNs can further introduce significant innovations to the error correction problem in deep learning hardware. Consider for starters that the final classification decision in a neural network is performed by finding the output position with the highest value in the last Softmax layer; an error in the intermediate variables is deemed therefore noncritical as long as the output position of the numerically largest value in the last layer remains constant. Second, the distribution of neural network parameters is observed to be clustered around zero and often span only a minute numerical range [8]. In a similar vein, not only are the majority of the observed activations restricted to small values as well, but they exhibit high levels of sparsity furthermore.

The inherent resilience characteristics and the predictable and well-behaved numerical distribution of neural network variables bolster the feasibility of approximate error correction. The value of an erroneous variable can be effectively estimated so as to minimize its impact on the output and maintain accuracy without necessitating perfect value restoration, thus significantly reducing the need for information redundancy for error correction operations. Furthermore, certain training techniques can increase the correlation across variables [9] and impart even higher precision to the value restoration process.

The outlined properties of neural networks are fundamental to the context of our discussion as they catalyze innovation in the neural network fault tolerance problem to deliver superior resilience goals often with insignificant overheads.

Learning error-checking invariants in neural networks

The flexibility of the training process can be utilized to inject useful invariants in DNNs, which facilitate highly cost-efficient error detection even across nonlinearities. We identify two types of potential invariants that can be injected into deep learning models for *coarse-grained detection* and *fine-grained error localization*. The embedding of these invariant types is achieved through dedicated regularization terms in the training loss function [10], and the integration of custom weight or activation propagation rules in the neural network graph [9], [11], respectively.

The first approach introduces coarse-grained error-checking invariants (i.e., checksums) in neural network layers by employing a custom regularization term in the training loss function. This scheme can be constructed easily in a few fundamental steps. We first partition the outputs of each neural network layer into two groups, such as two neuron groups for the fully connected layers or two output channel groups for the convolutional layers. The outputs of the two partitions are accumulated separately in the channel dimension, and the mean squared difference of the accumulations for the two partitions is included as a penalty term in the loss function. The introduced penalty term reduces the observed maximum mean squared differences between the group summations (i.e., maximum checksum deviation) to the tune of a few magnitude orders, as demonstrated in Figure 1. The significant reduction in group differences and the consequent balancing provides ample resolution for error detection in the case of a numerical error modifying the sum in one partition, thus distorting the established balance. Fortuitously, the employed penalty term of group differences usually operates as a regularizer as well, leading to improved generalization and higher test/validation set accuracy, as also visualized in Figure 1.

Error checking at runtime can be performed by monitoring the group summation differences at each layer through a single additional neuron or a 1×1 convolution filter in the subsequent layer, as an example for the fully connected layers illustrates in Figure 2. The expected maximum deviation value in the absence of errors is identified through one-time

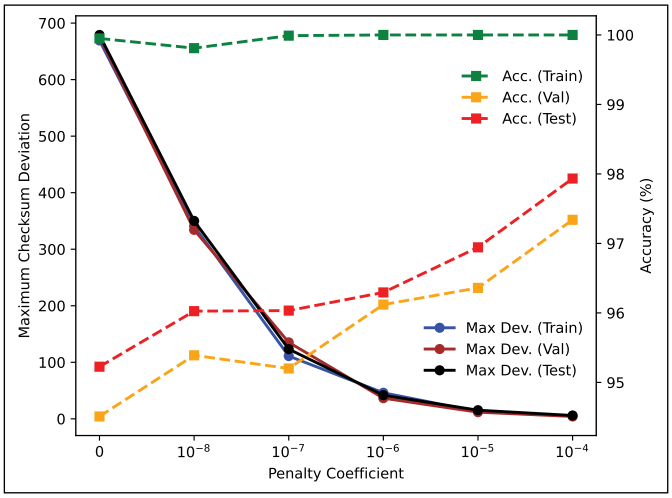


Figure 1. Balance checksum deviation and model accuracy due to regularization (DNN model similar to AlexNet trained on German Traffic Sign Recognition Benchmark [GTSRB] data set).

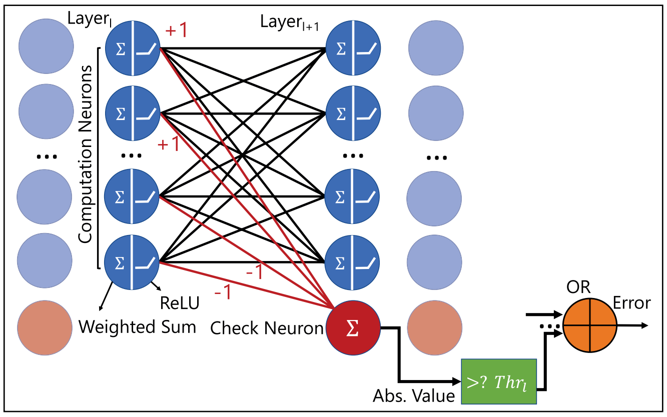


Figure 2. Error checking at inference time via coarse-grained invariants.

profiling at each layer. If the output of the checker neuron deviates from zero by more than the profiled threshold for the corresponding layer, it signals an error presence. Our experimental analysis in [10] affirms that balanced output partitions detect more than 95% of the error-caused misclassification cases without any false alarms. The software implementation of this approach incurs a negligible increase in parameters and a performance overhead constrained to around 1% on a CPU.

While such coarse-grained invariants provide information only on error presence but no precise error localization, they enjoy the advantage of an entire software implementation with native neural network operations that are deployed with ease in off-the-shelf hardware platforms, including CPUs, GPUs, and deep learning accelerators. Moreover, the footprint on the obtained model is found to be rather minimal. Coarse-grained invariants are ideal for safety-critical designs that necessitate the detection of rare error incidents (i.e., single-event upsets caused by high-energy particles) where the mitigation action can be effected through system-level approaches.

An alternate approach we propose brings on more precise localization of anomalous DNN variables through the imposition of local variable propagation rules in the computational graph. An example propagation rule could be a simple numerical order relationship across the neighboring variables where a weight or activation variable A_c is declared anomalous if its magnitude unusually exceeds the neighboring value A_{c+1} by a preset relationship (Figure 3a), otherwise deemed normal (Figure 3b) and propagated in the neural network graph with no modification.

Fine-grained invariants can be embedded into neural network layers by imposing custom propagation rules in both the forward and the backward pass of training and ensuring that the deep learning model attains a competitive accuracy within the confines of these rules. Invariant integration incurring neither additional information redundancy nor baseline accuracy degradation can be achieved through inherent model redundancy and training process flexibility [9], [11].

The violation of these embedded invariants due to a hardware error can localize the erroneous variables at inference time with high precision. The fine-grained nature of such invariants engenders precise error localization even in the presence of multiple errors. After error localization, the novel error suppression methods to be outlined in the next section can be employed to contain error effects and maintain neural network accuracy gracefully even while suffering extreme bit error rates unimaginable in conventional fault-tolerant designs.

While the footprint of fine-grained invariants on the trained model is more noticeable than the coarse-grained counterparts, the inherent redundancy of modern deep learning models allows injection of such invariants into the model without requiring any additional information redundancy or impacting error-free model accuracy. Minor hardware extensions are necessary for the accelerator designs to check the invariant conditions and perform the mitigation actions efficiently, as discussed further in the next section. Unlike their coarse-grained counterparts, fine-grained invariants can localize the errors with high precision, and when paired with novel error suppression methods, deliver complete algorithmic resilience for even extreme bit error rates of up to a few percent.

Maintaining neural network accuracy with error suppression

We have illustrated that the problem of error detection could be resolved in an innovative manner by integrating computational invariants into neural networks and employing them for error detection in inference. The localization of the errors through the fine-grained invariants can be followed up by the suppression of the numerical distortion prior to the execution of each layer. Such suppression can be effected by dropping variables (setting to zero), clipping their magnitude to within the usual range [11], or passing the variables through filtering operations [9], as demonstrated in Figure 4. Disproportionate error effects can thus be arrested at their tracks and reduced back to size before they have had a chance to diffuse in the network. The beneficial impact of error suppression can also be enjoyed when dropping erroneous variables by setting them to zero. This alternate approach can effectively alleviate the impact of large-magnitude errors since the inherent sparsity of DNNs and the clustered distribution of DNN variables around zero leads to graceful toleration of variable drop effects.

Introduced fine-grained invariants can be checked and the anomalous variables suppressed efficiently at inference time through dedicated hardware extensions in DNN accelerators. These extensions can be implemented at minimal hardware cost with basic hardware components such as comparators and multiplexers. Figure 5 demonstrates a possible option for hardware integration where the detection and suppression are carried out prior to processing each neural network layer in the accelerator pipeline. The proposed placement of the dedicated hardware unit protects against SRAM buffer errors and timing errors in the computational fabric. Anomaly detection and suppression techniques require no additional information redundancy, with the associated operations performed efficiently in hardware with no throughput impact while imposing area and power overheads of less than 0.5% and 0.2%, respectively, when implemented on a typical DNN accelerator (DNNWeaver v2.0 [12]).

Overall, the two outlined novel mechanisms of error localization and suppression can be coupled to deliver highly resilient neural network processing systems. Potential error locations are pinpointed

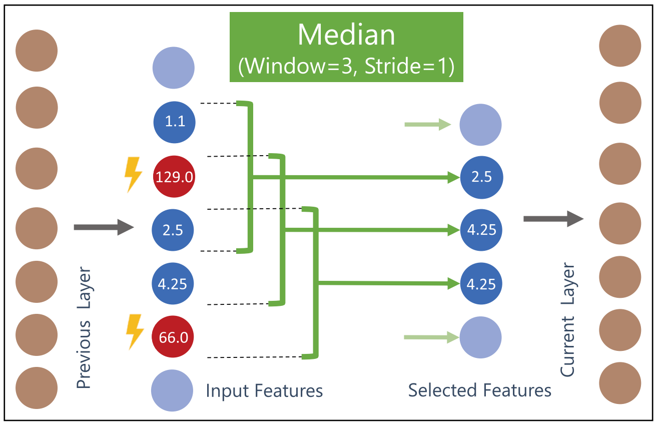


Figure 4. Median filtering in fully connected layers.

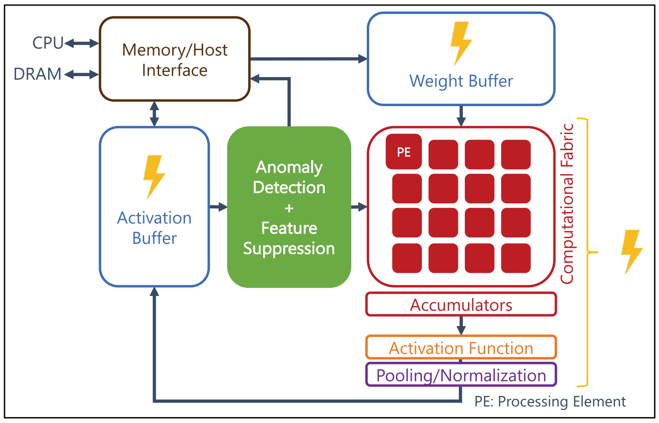


Figure 5. Anomaly detection and suppression in a deep learning accelerator.

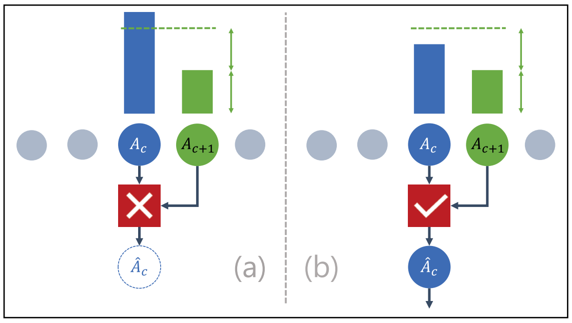


Figure 3. Anomaly detection with local magnitude comparison.

through computational invariants injected through the learning process, and anomalous variables are snapped back outright before they have had a chance to propagate and influence the neural network decisions. In contrast to the outright effect of the large deviation bit errors encountered, the impact of suppression, which accords with the inherent distribution of neural network variables, on model accuracy is highly muted. The described approach is a potent strategy to engender approximate error resilience methods, revolutionizing our perspectives on functional safety for deep learning hardware. As a result, strict safety goals can be attained at minimal additional cost in DNN applications.

Experimental analysis

We demonstrate the effectiveness of the proposed methods in Figure 6 by performing error injection on three different neural network and data set pairs and measuring the classification accuracy at various activation and weight error rates.

Baseline results establish the inherent extent of the fault tolerance of the deep learning models with no embedded error resilience mechanism. DNNs with *median filters* check activations prior to the execution of each layer through filtering operations in the channel dimension. We implement the proposed median filtering technique with two distinct filter sizes to measure its effectiveness at different design points. We construct *fine-grained invariants* in neural network models through the local magnitude comparison method (Figure 3) and perform anomaly suppression on erroneous variables by *dropping* outright or *clipping* their magnitude to the expected range. Finally, we construct two triple modular redundancy (TMR) techniques for comparison. The target neural network is replicated three times, each instance executed independently to deliver its own classification, and the final decision is made through majority voting in a *model-based TMR* technique. In *layer-based TMR*, each neuron/filter is triplicated, and the neuron/filter output is produced through majority voting at each layer before proceeding with the execution of the subsequent layer.

Our experimental analysis validates our expectations that deep learning models are prone to a noticeable accuracy drop even at low error rates when suffering large-magnitude error deviations. We

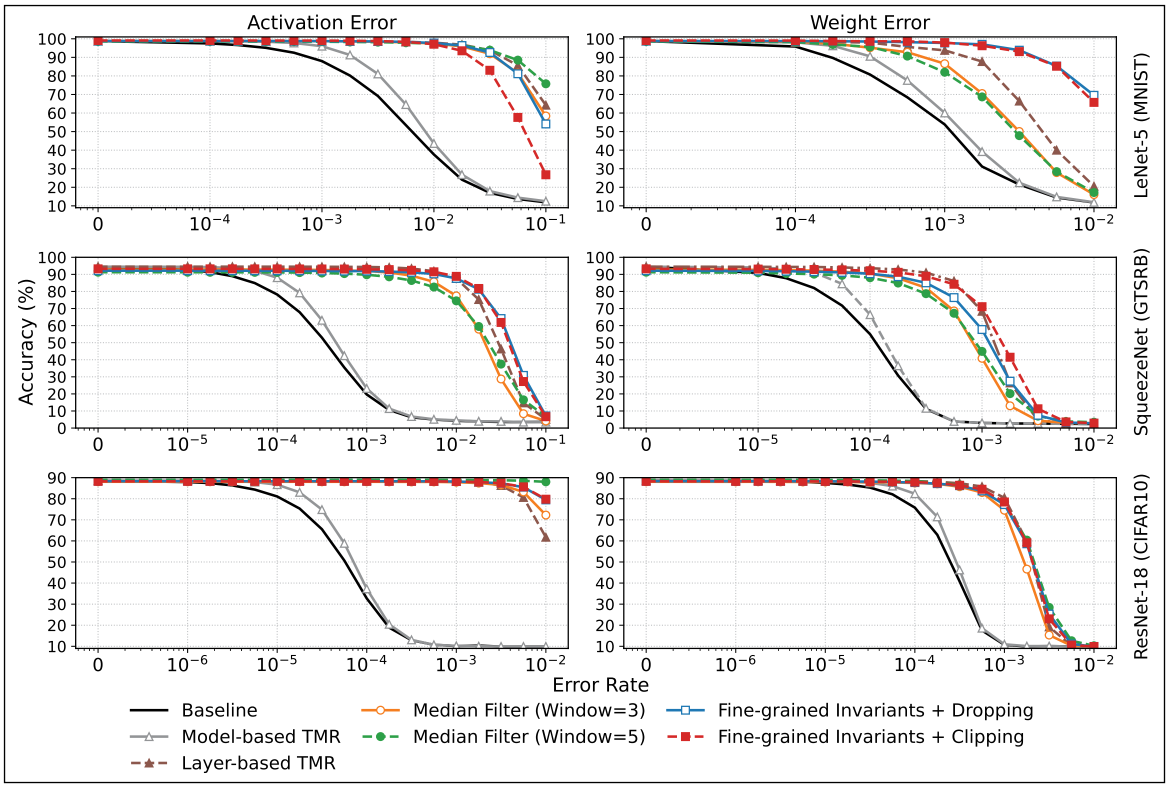


Figure 6. Error resilience of DNNs under different fault tolerance methods.

observe that model-based TMR is often ineffective in boosting neural network resilience over baseline models. Layer-based TMR, on the other hand, is shown to be effective as each bit error is squashed immediately at each layer in this scheme before it has had a chance to propagate to neural network outputs. However, the large overheads of layer-based TMR may prove inhibitive in practical applications.

The proposed novel techniques attained through median filtering and fine-grained invariants coupled with error suppression (i.e., dropping and clipping) can provide extensive bit error resilience in the target deep learning models even when afflicted by thousands of bit errors scattered throughout the model. The proposed methods operate on activations and offer resilience against up to 10,000 times higher activation error rates; an indirect impact on weight error resilience to withstand up to 50 times higher error rates can be furthermore enjoyed as the propagation of weight errors in a particular layer can still be suppressed through the activation checks in the subsequent layer. Weight error resilience can be further bumped up by performing invariant injection and error suppression operations on the weights directly prior to each layer's execution. We observe that neural network error resilience characteristics obtained through the proposed novel techniques are even superior to layer-based TMR despite incurring no additional information redundancy.

Discussion

The novel techniques we outline focus primarily on the data path and buffers where the majority of hardware resources are allocated, and the cost of delivering functional safety through conventional fault tolerance methods proves to be exceedingly high. While control path integrity is just as important, the inordinate cost of traditional techniques can be easily borne for the small footprint of control circuitry in deep learning accelerators that may necessitate absolute resilience.

We attain strong error resilience and competitive accuracy through the inherent flexibility and redundancy of neural networks. Neural networks embed redundancy in various dimensions, and the redundancy types that cannot be effectively squeezed through model compression can be utilized for boosting error resilience at no additional cost. Our preliminary investigations indicate that model compression methods such as pruning can be applied to the proposed models without impacting their outstanding error resilience characteristics. On the other hand, introduced invariants can lead to dependencies across neighboring variables, which need to be taken into consideration during the model compression process. Future investigations will focus on the effective deployment of model compression techniques on the proposed models.

While the outlined analysis focuses on convolutional and fully connected layers, the proposed techniques are expected to generalize to a wide range of DNNs, such as recurrent models and other emerging neural network architectures.

Overall, the computational characteristics of neural networks can enable significant breakthroughs for the error resilience problem in deep learning hardware, delivering highly effective solutions at imperceptible overheads.

The error resilience of DNNs can be boosted noticeably by restricting and containing the numerical contribution of the errors without necessitating explicit error correction steps. The proposed novel error detection and remediation techniques can complement each other seamlessly to tackle errors with high precision, while neither incurring additional information redundancy nor having a noticeable impact on the error-free classification accuracy. The proposed approach innovatively redefines the error resilience problem in the context of DNNs, thus unlocking effective opportunities for efficiently embedding functional safety into the next generation of machine intelligence hardware. ■

References

[1] E. Ozen and A. Orailoglu, "Sanity-check: Boosting the reliability of safety-critical deep neural network applications," in *Proc. ATS*, 2019, pp. 7–12.

[2] G. Li et al., "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," in *Proc. SC*, 2017, pp. 1–12.

[3] S. Pandey, S. Banerjee, and A. Chatterjee, "Error resilient neuromorphic networks using checker neurons," in *Proc. IOLTS*, 2018, pp. 135–138.

[4] S. Pandey, S. Banerjee, and A. Chatterjee, "ReiNN: Efficient error resilience in artificial neural networks using encoded consistency checks," in *Proc. ETS*, 2018, pp. 1–2.

[5] L.-H. Hoang, M. A. Hanif, and M. Shafique, "FT-ClipAct: Resilience analysis of deep neural networks and

improving their fault tolerance using clipped activation," in *Proc. DATE*, 2020, pp. 1241–1246.

[6] G. B. Hacene et al., "Training modern deep neural networks for memory-fault robustness," in *Proc. ISCAS*, 2019, pp. 1–5.

[7] A. Chaudhuri et al., "C-testing of AI accelerators," in *Proc. ATS*, 2020, pp. 1–6.

[8] S. Han et al., "Learning both weights and connections for efficient neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 1135–1143.

[9] E. Ozen and A. Orailoglu, "Boosting bit-error resilience of DNN accelerators through median feature selection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 11, pp. 3250–3262, Nov. 2020.

[10] E. Ozen and A. Orailoglu, "Concurrent monitoring of operational health in neural networks through balanced output partitions," in *Proc. ASP-DAC*, 2020, pp. 169–174.

[11] E. Ozen and A. Orailoglu, "Just say zero: Containing critical bit-error propagation in deep neural networks with anomalous feature suppression," in *Proc. ICCAD*, 2020, pp. 1–9.

[12] H. Sharma et al., "From high-level deep neural models to FPGAs," in *Proc. MICRO*, 2016, pp. 1–12.

Elbruz Ozen is pursuing a PhD with the University of California at San Diego, La Jolla, CA 92093 USA. His research interests focus on robust and resource-efficient hardware architectures for machine intelligence. Ozen has an MS in computer engineering from the University of California at San Diego. He is a Graduate Student Member of IEEE.

Alex Orailoglu is a professor of computer science and engineering at the University of California at San Diego, La Jolla, CA 92093 USA. His research centers on the establishment of robust and reliable computational structures. Orailoglu has a PhD in computer science from the University of Illinois at Urbana–Champaign, Urbana–Champaign, IL, USA. He is a Golden Core Life Member of IEEE.

■ Direct questions and comments about this article to Elbruz Ozen, Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA 92093 USA; elozen@eng.ucsd.edu.

Fault-Tolerant Neural Network Accelerators With Selective TMR

Timoteo García Bertoa
Xilinx Research Labs, Dublin, Ireland
Queen's University Belfast, Belfast BT7 1NN, U.K.

John McAllister
Queen's University Belfast
Belfast BT7 1NN, U.K.

Giulio Gambardella, Nicholas J. Fraser, and Michaela Blott
Xilinx Research Labs, Dublin, Ireland

Editor's notes:
This article presents a tool that analyzes sensitive computations in the neural network and triplicates them to increase the functional safety of the neural network accelerator.

—Fei Su, Intel Corporation

The fast growth of novel embedded heterogeneous architectures in the last decade opened doors to machine learning acceleration targeting various fields and applications, some of which are categorized as safety-critical, such as self-driving cars, flight-control systems, or aerospace. The supervision and diagnosis of faults in these kinds of applications are of great importance to prevent malfunctions or total system failure, hence requiring a high level of reliability.

A common approach to achieving fault tolerance in hardware is the use of triple modular redundancy (TMR), performing a majority vote on triplicated modules. However, machine-learning accelerators are extremely power- and resource-hungry, making TMR an undesirable solution, due to a 200% hardware overhead. It should also be considered that not all portions of a neural network (NN) have the same effect on the overall accuracy, thus exploring in detail the inherent feature of NNs enables more fine-grained approaches. In this work, we explore applying redundancy only to the most critical elements within the NN hardware accelerator, opening a new design space exploration to reduce resource usage while offering a certain degree of fault tolerance.

This article presents selective TMR (STMR), an automated tool to export fault-tolerant NNs using field-programmable gate array (FPGA)-based hardware. The main features of this tool are:

- It applies redundancy to computations that, in the presence of single faults, have a greater impact on the overall accuracy of the network, achieving a fault tolerance for a highly reduced hardware cost.
- It preserves the throughput of the NN accelerator, with a negligible increase in latency.
- It offers configurable fault tolerance by setting a minimum requirement of accuracy in the presence of a single fault, providing control over the degree of redundancy applied to the NN.
- It provides layer-level information on single-fault detection to the host, for further fault correction at the system level (e.g.,scrubbing).

The following sections introduce related work and required background leading to an STMR

Digital Object Identifier 10.1109/MDAT.2022.3174181
Date of publication: 11 May 2022; date of current version: 10 March 2023.

model, an explanation of the tool flow, and detailed analysis based on a use case.

Related work

Prior works aiming to improve the reliability of NN inference fall into the following categories:

1) Hardware redundancy, such as TMR.
2) Numerical redundancy, such as error correction code (ECC) and algorithm-based fault tolerance (ABFT).
3) NN training for robustness, such as fault-aware training (FAT) [1].

These approaches are often used in combination to create a reliable system. The approach described in this article is a variant on 1), while presenting results when 3) is also applied. However, before discussing prior works' methods to improve the robustness of NN accelerators, it is worthwhile to review works that attempt to analyze the reliability of NNs *without* any particular fault mitigation techniques being applied.

In particular, Gambardella et al. [2] presented an error injection methodology to evaluate fault tolerance in NNs. They observed that single faults onto single neurons at specific locations in convolutional layers can cause accuracy drops of up to 10%. To mitigate this, the authors propose an STMR scheme and provide an analysis of potential resultant reliability and hardware cost based on models. Similarly, Libano et al. [3] explored how the reliability of NN accelerators changes with various network and accelerator parameters. They show that reliability improves with reduced data precision and increased parallelism. More recently, Xu et al. [4] analyzed the probability of system exceptions of FPGA-based accelerators due to hardware faults. The authors found that system exceptions can dominate the reliability of the system and evaluated full TMR as a potential solution. In addition to the works above, Brosser et al. [5] showed that periodic scrubbing (reprogramming) is vitally important when FPGAs are deployed in low earth orbit (LEO) or geostationary earth orbit (GEO) satellites.

Closer to this work, several works have proposed methods to apply full or partial TMR to NN accelerators. For example, Wang et al. [6] implemented full TMR on a custom lightweight convolutional NN (CNN) topology. Their approach significantly improved the error rate (33.59% error rate reduction) although they incur a large increase in hardware resources. As an example of partial TMR, Libano et al. [7] proposed selective hardening, where they triplicated the most sensitive layers and validated the approach by means of neutron radiation testing, showing high fault-masking (40%) with marginal hardware overhead (8%). SHIELDeNN [8] also proposed a framework applying partial TMR to the weights within sensitive NN layers. The authors showed an improvement in error resilience while incurring only a little hardware overhead. Finally, Spyrou et al. [9] employed both partial TMR (to the output layer of the NN) and a training technique to improve the robustness of spiking NNs (SNNs).

Zhao et al. [10] is an example of ABFT. The authors proposed four different ABFT schemes that protect against single-event effects (SEEs) in convolutional layers. Together, the schemes provide effective fault tolerance, while incurring little (8%) runtime overhead.

In this work, we propose fault-tolerant NN accelerators leveraging parallelism using FPGAs for acceleration (as per suggested in [3]), targeting convolutional layers (as [10]), targeting reduced hardware overhead (as suggested in [4] and [6]), and applying partial TMR as in [7]–[9], but with a finer-grained approach to selective hardening or SHIELDeNN, as we analyze and triplicate individual channels *within* a layer, instead of the entire layer. This work could be considered to be an extension of Gambardella et al. [2], developing the ideas proposed in the work into a tool, which generates reliable accelerators.

Background

STMR builds on three main prior works: 1) FINN [11]; 2) error injection [2]; and 3) FAT [1]. In particular, we utilize the error injection campaign proposed by Gambardella et al. [2] with the specific aim to identify sensitive computations within the NN. We also extend FINN to support detecting and correcting single faults on selected output channels. Finally, we leverage FAT to compare the effectiveness of STMR on quantized NNs (QNNs) which are trained to be resilient to faults to networks trained with standard training (SAT) techniques.

FINN framework

FINN [11] is an end-to-end framework that enables the deployment of QNNs into FPGA-based hardware platforms, especially focusing on extremely reduced precisions down to binary NNs (BNNs), leveraging heavily quantized weights and activations enabling their storage into ON-chip memory and reducing the compute hardware cost. The architecture of machine learning accelerators built using FINN consists of a sequence of layers with dedicated processing elements (PEs), offering a high level of configurability to target-specific throughput for a submicrosecond latency. The FINN toolchain allows generating dataflow architectures configured for different low precisions, network topologies, or data sets.

For the scope of STMR, it is important to remark that FINN-based accelerators can be deployed onto FPGAs without an accompanying CPU, thus requiring redundancy to be implemented during inference rather than at the algorithmic level. For TMR implementation in FINN, special attention is put on the structure of the convolutional layer.

Each convolutional layer in FINN receives an input feature map (IFM) and produces an output feature map (OFM). The IFM is first transformed into an input matrix by a sliding window unit (SWU), which *lowers* the convolution to a matrix multiplication on the fly, by means of im2col. From here, the matrix multiplication is mapped to the main computational primitive in FINN: the matrix-vector threshold unit (MVTU). The MVTU calculates matrix-vector products, and the subsequent quantized activations by means of thresholding.

The OFM number of channels (C_{OFM}) for each layer is determined by the number of filters used during convolution, and the level of parallelism to compute simultaneous OFM channels is determined by the number of PEs utilized within the layer. If the number of PEs, N_{PE}, is less than the number of OFM channels, that is, $N_{\mathrm{PE}} < C_{\mathrm{OFM}}$, then each PE computes multiple output channels. The amount of OFM channels each PE computes is known as the neuron folding factor and is given by $F_n = C_{\mathrm{OFM}}/N_{\mathrm{PE}}$, where $C_{\mathrm{OFM}}\%N_{\mathrm{PE}} = 0$. The specific channels that the ith PE in the MVTU computes are given by

$$C_{\mathrm{PE}i} = (i + j\,N_{\mathrm{PE}})_{j=0,\dots,F_n-1}. \qquad (1)$$

This property plays an important role in determining the schedule described in the following section.

Error injection methodology

Error injection is crucial to understanding how the NN behaves in the presence of faults and how different faults impact the overall accuracy. Gambardella et al. [2] presented an error injection methodology to evaluate the fault tolerance of NNs. This method consists of altering threshold values utilized in the MVTU at run time and injecting activation values in the next layer. For instance, if thresholds are set to a maximum or minimum value for a single channel of a layer in a BNN, the corresponding activations are forced to a permanent value of 0 or 1. Evaluating different channels and layers in each iteration, the overall accuracy of the network is computed and reported, obtaining information on the sensitivity against single faults for all targeted error models. It is worth noticing how the sensitivity analysis is specific to a trained NN, and a new evaluation is needed if the same NN is retrained or if the topology changed. This error injection methodology was adopted to apply the channel stuck at the error model to obtain channel fault tolerance analysis reports. For those readers who wish to further understand the error model and error injection framework, we refer them to Gambardella et al. [2].

FAT for reliable inference

The STMR approach presented in this article is independent of the process of training NNs. However, the selection of the most critical portion of the NN to be triplicated heavily depends on the trained parameters and the results of the error injection.

In this context, the FAT methodology proposed by Zahid et al. [1] introduces a new error injection layer component in the network definition, enabling error models to be utilized as a part of the training process. The use of FAT for training improves the resilience of the network, proving greater tolerance than networks trained with SAT techniques for different error models and precisions, with higher error-free accuracy and higher minimum accuracy in the presence of faults. In this article, the use-case shown considers the utilization of trained parameters obtained using both SAT and FAT methodologies, to assess the implications of the training methodology to the hardware resource usage of the STMR accelerator. However, FAT is not a requirement for STMR, but rather an additional resource to seek fault tolerance.

STMR modeling

TMR mapping

The mapping of convolution computations to the MVTU in FINN is *output stationery*, meaning that each

PE computes all outputs on one or more entire channels in an OFM. In the case where a PE calculates more than one output channel, it is imperative that each triplicated channel in a triplet is calculated on a separate PE, to avoid a single point of failure. This is achieved in STMR by ensuring that the replicated channels in the triplet are each mapped to a unique PE, assuming there are three or more PEs. Specifically, this is achieved by placing the replicated channels immediately after the position of the original channel. This is shown in Figure 1a, where three of the six channels (in positions 0, 2, and 4) are identified as being sensitive to faults. After STMR mapping, the number of channels increases to 12, where channel 0 turns into a triplet of channels in positions 0, 1, and 2; channel 2 into a triplet with positions 4, 5, and 6; and channel 4 into a triplet with positions 8, 9, and 10.

TMR majority vote

The majority vote is implemented by an additional hardware component placed right after the MVTU or convolutional layer in FINN, named TMR check (TMRC). TMRC receives an OFM with triplicated channels as input and outputs an OFM with valid results, as well as providing status information to the host through two flags: 1) an error-detected flag and 2) an error-corrected flag. In essence, this unit performs a comparison for each triplet. Three possible scenarios could occur during the process, each case represented in Figure 1b with triplets T1, T2, and T3, respectively:

- All three channels are identical. In this situation, the result is assumed as valid and forwarded to the output, with no errors flagged.
- Two channels are identical and one has a different value. In this situation, the value present twice is considered valid and forwarded to the output, thus detecting and tolerating the single error. The error-detected and error-corrected flags are also raised.
- All three channels are different. In this situation, the result of the first channel is selected by default as valid and forwarded to the output. The error-detected flag is raised, but the error-corrected flag is not.

Additionally, TMRC is configurable to define input and activation precisions, the number of triplicated channels, OFM dimensions, or redundancy factor, assumed as three in this work.

STMR tool flow

The STMR tool aims to automatically generate a hardware accelerator that achieves the desired level of single-fault tolerance.

To do so, our codesign flow first triplicates parameters of explored sensitive channels of each layer (software), second mapping these, leveraging high-level synthesis (HLS) for the model transformations (hardware). This redefines the NN model, with the advantage of computation-level redundancy for a low cost in resources.

As illustrated in Figure 2, the tool requires as input:

- Trained parameters, including the complete NN topology definition. It should be noted how there is no restriction on the training methodology adopted, and training is completely independent of STMR implementation.

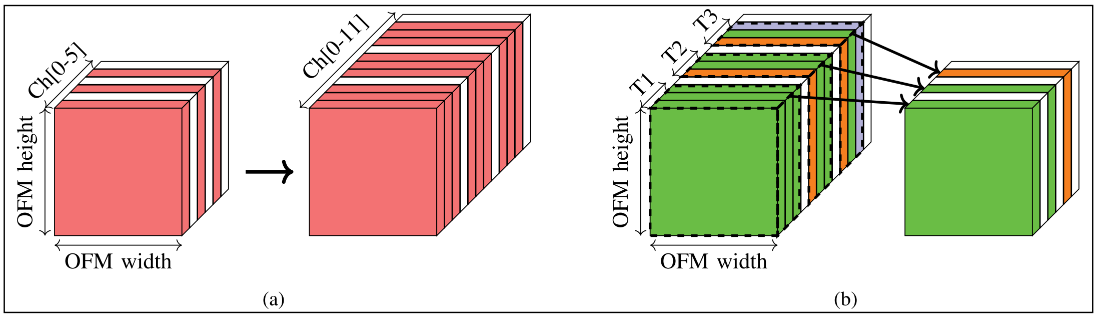


Figure 1. STMR mapping and majority vote. (a) TMR PE mapping: critical channels 0, 2, and 4 are triplicated, placing replicas in a consecutive manner. Red represents critical channels. (b) STMR majority vote for each triplet (T1, T2, T3). Green represents expected values (same for each triplet), whereas orange and blue represent non-expected values (possibly due to faults).

- A sensitivity report, as a result of the error injection campaign described in the previous section performed on the target NN.
- A minimum accuracy requirement in the presence of single faults, which the user can specify depending on the target fault tolerance requirement. Redundancy is regulated as a consequence of minimum accuracy choice, which aims to act as a fault-tolerance guarantee, limited by the FPGA capacity.

Given the inputs, an automated process exports the fault-tolerant NN accelerator, relying on a series of Python scripts included in FINN and the HLS library which leverage the Xilinx toolchain to build and produce the final hardware accelerator. Figure 2 illustrates the steps of this process, which starts with a Python-based three-step procedure:

- Analysis of the sensitivity report obtained from the error injection stage, selecting which channels of each layer will be triplicated based on the required minimum accuracy in the presence of faults.
- Validation of the parallelism, namely the number of PEs used for each layer of the network, ensures the TMR mapping described in the previous section is viable for the list of critical channels previously selected.
- Automatic export of the weights and parameters, including the triplication of the values corresponding to each triplicated channel.

The process continues with the generation of the QNN accelerator itself. The FINN HLS components are generated for all layers within the network, with some layers augmented with triplicated channels and a TMRC layer (described in the previous section) placed after them. Once the HLS is generated for the fault-tolerant network, Vivado HLS synthesizes it, exporting an intellectual property (IP) block which can be integrated and built within a block design in Vivado.

Fault-tolerant networks with STMR

The STMR tool has been tested and validated on a set of QNNs inspired by BinaryNet [12], which consists of six convolutional layers, two max pool layers, and three fully connected layers, is called CNV and first proposed by Umuroglu et al. [11]. The CNNs have been trained on the CIFAR-10 data set [13] to classify images among 10 classes with reduced precisions and are referred to as CNVW1A1, CNVW1A2, or CNVW2A2, where the number after W and A are the bitwidths of weights and activations, respectively.

STMR use-case

Let us consider an SAT-trained CNVW1A1 network whose computed error-free accuracy is 84.46%. The results of the error injection campaign performed over this network to analyze its fault sensitivity are collected in Table 1, where clearly the convolutional layers (0–5) experience a higher accuracy drop when compared to the fully connected layers (6 and 7). More specifically, layer 1 experiences a minimum accuracy of 59.28% when a single channel was stuck at 1.

Setting the STMR minimum accuracy requirement to 83% and using the automated flow to generate a fault-tolerant accelerator, it requires 16, 50, 92, 56, and 7 triplicated channels in the first five convolutional layers. This leads to a total of 221 triplications (≈12% of the total channels in the network) and the implementation of one TMRC layer per each convolutional layer containing redundancy.

The same CNVW1A1 network trained using FAT gives a computed error-free accuracy of 84.8%. The minimum accuracy observed during error injection is 81.08%. Targeting 83% minimum accuracy, the STMR tool exports a fault-tolerant NN with only 1, 36,

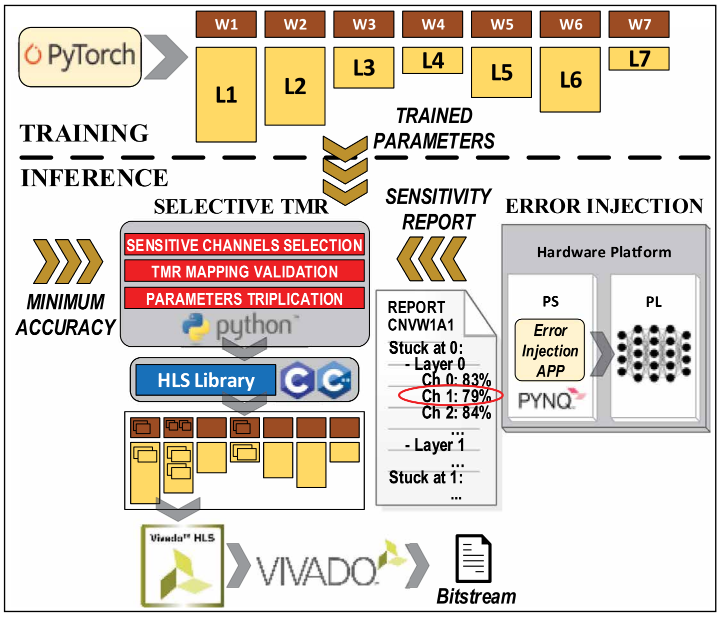


Figure 2. General overview of STMR tool flow.

Table 1. Error injection campaign and STMR validation for 83% minimum accuracy.

CNVW1A1 trained with SAT - Error-free accuracy: 84.46%						
			Error model: channel stuck at 0		Error model: channel stuck at 1	
Layer	Channels	STMR Channels	ERROR INJECTION: minimum accuracy [%]	STMR VALIDATION: minimum accuracy [%]	ERROR INJECTION: minimum accuracy [%]	STMR VALIDATION: minimum accuracy [%]
0	64	96	81.74	83.31	81.15	83.43
1	64	164	78.70	83.18	59.28	83.06
2	128	312	77.87	83.10	79.05	83.06
3	128	240	83.08	83.08	76.40	83.01
4	256	270	82.61	83.04	82.52	83.00
5	256	256	84.34	84.34	84.26	84.26
6	512	512	84.33	84.33	84.34	84.34
7	512	512	84.40	84.40	84.40	84.40

2, and 6 triplicated channels in the first four convolutional layers, respectively, for a total of 45 triplicated channels (≈2% of the total).

These results show a greater benefit when training the network using FAT, reducing considerably the redundancy required for the same minimum accuracy target.

Identical steps were followed to perform error injection and export CNV networks with other accuracy requirements and precisions, whose cost in hardware will be addressed in the following sections.

Validation, throughput, and latency

To validate the functionality of STMR, an error injection campaign was performed into STMR-exported networks. The campaign consists of injecting errors to each individual channel of each layer, including the triplicated ones which will be treated exactly like the original. The campaign aims at confirming that the triplication performed by the STMR tool and implemented in the hardware accelerator is able to guarantee error-free accuracy in the presence of faults in one of the triplicated channels. When injecting single faults into such channels, the majority vote will tolerate the fault and the computed accuracy will be equal to the error-free accuracy, while the error detection and error correction flags will be raised by the hardware accelerator.

For instance, the result of this error injection campaign for the network example in the previous section, which had implemented STMR for a minimum accuracy of 83%, is shown in Table 1, where, compared to the error injection results, clearly the minimum accuracy for the first five convolutional layers is kept above the 83% threshold, thus validating the STMR implementation.

The parallelism of STMR networks cannot be exactly preserved, as the number of PEs varies depending on the number of triplications and folding requirements as explained in the previous section. However, having control over the parallelism for each convolutional layer for the previous example, a throughput of ≈21 k frames/s is preserved for both SAT and FAT trained networks, while leveraging the benefits of STMR. Latency experiences a slight increase due to the inclusion of TMRC layers in the network. This increase is negligible, as TMRC has been fully unrolled to perform majority vote for one pixel in one clock cycle. In the case of SAT-STMR-CNVW1A1, the addition of 221 triplications produces a latency increase from 142.3 to 145.2 μs, whereas the addition of 45 triplications for the FAT-STMRCNVW1A1 case causes an increase from 142.3 to 143.3 μs.

Hardware resource analysis

Area savings of STMR is one of the key reasons and motivations of this research, enabling designers to explore the design space of hardware usage versus fault tolerance. The hardware usage increase when applying STMR to a FINN accelerator can be divided into two main components: 1) majority voter (TMRC) and 2) increased PEs to compute triplicated channels. TMRC only requires look-up tables (LUTs) and flip-flops (FFs), with resource usage linearly dependent on activation precision and number of OFM channels (including triplications).

In all our examples, the resource usage of TMRC was less than <7.5% of the resources of the

Table 2. STMR hardware cost when targeting <2% accuracy drop and throughput of 21 k frames/s.

	Network: CNVW1A1						Network: CNVW1A2						Network: CNVW2A2					
	NO TMR		FULL TMR		STMR		NO TMR		FULL TMR		STMR		NO TMR		FULL TMR		STMR	
	SAT	FAT	SAT	FAT	SAT	FAT	SAT	FAT	SAT	FAT	SAT	FAT	SAT	FAT	SAT	FAT	SAT	FAT
Measured single-fault minimum accuracy [%]	59.28	81.08	84.46	84.80	83.00	83.03	64.50	85.24	87.49	88.19	87.00	87.00	68.00	86.69	88.24	88.57	87.01	87.19
Targeted single-fault minimum accuracy [%]	-	-	84.46	84.80	83.00	83.00	-	-	87.49	88.19	87.00	87.00	-	-	88.24	88.57	87.00	87.00
Number of triplications	-		1920		221	45	-		1920		592	115	-		1920		408	2
CLB LUTs $[x10^3]$	45.8		≥137.4		71.5	55.2	81.1		≥243.4		200.7	135	94.1		≥282.2		195.7	95.1
CLB Registers $[x10^3]$	60.6		≥181.9		91.9	70.5	84.7		≥254		181.8	124.3	83.4		≥250.2		141.8	85.1
36kb BRAM	147.5		≥442.5		210.5	147.5	146.5		≥439.5		334	210	333.5		≥1000.5		665	339.5
LUTs increase [%]	-		≥200		56	21	-		≥200		147	66	-		≥200		108	1
BRAM increase [%]	-		≥200		43	0	-		≥200		128	43	-		≥200		99	2

corresponding MVTU. Compared to the resource overhead of triplicating several OFM channels, this is relatively low.

The majority of hardware cost increase depends on the compute part as well as BRAM usage increase due to partial triplication of parameters. Considering as baseline the CNVW1A1 example, LUTs, FFs, and 36-kb BRAM utilization are 45.8 k, 60.6 k, and 147.5, respectively. Hence, a full TMR solution would require ≥137.4 k LUTs, ≥181.9 k FFs, and ≥442.5 BRAMs. However, if we leverage the STMR tool to export a fault-tolerant CNVW1A1 network trained using SAT targeting a minimum accuracy of 83% in the presence of faults, resource utilization is 71.5 k LUTs, 91.9 k FFs, and 210.5 BRAMs. This means an additional cost of 56% in LUTs, compared to the theoretical ≥200% for full TMR, while guaranteeing a maximum drop in accuracy of <1.5%, increasing by 23.72% the minimum accuracy under single faults.

When using the FAT trained NN with the same target accuracy in the presence of faults, we experienced a resource utilization of 55.2 k LUTs and 70.5 k FFs, resulting in increased utilization of 21% in LUTs, compared to the theoretical ≥200% for full TMR and 56% for SAT-STMR. Additionally, optimization of BRAM utilization mapped to LUTs used as memory is also leveraged by the synthesis tools, as in this case only 147.5 BRAMs were utilized.

The full set of results is collected in Table 2, presenting the overall hardware cost for CNVW1A1, CNVW1A2, and CNVW2A2 accelerators implemented with STMR. The target accuracy drop in the presence of single faults has been set to be less than 2% for all precisions, and the results have been reported using both SAT and FAT training methodologies. The reader can observe great area savings when comparing the hardware cost of a full TMR approach versus the STMR solution proposed. Also, it can be observed that the error-free accuracy and hardware cost increase with higher precisions.

The complete design space of hardware cost versus worst-case classification error (i.e., 1—minimum accuracy under a single fault) against utilization in LUTs is shown in Figure 3, where full TMR solutions marked in the graph clearly offer the minimum worst-case error, but for the maximum hardware cost, and solutions without redundancy show a high worst error case which does not guarantee high levels of accuracy in the presence of faults. The intermediate results define the design space for STMR fault-tolerant solutions considering different minimum accuracy requirements, where the hardware cost is highly reduced. The most attractive solutions shown in this figure respond to the intermediate data points for FATSTMR-CNVW1A1, FAT-STMR-CNVW1A2, and FAT-STMRCNVW2A2 generated networks, being all Pareto dominant to the SAT counterparts.

In this work, we presented STMR, an automated tool to generate fault-tolerant NNs for machine-learning accelerators in FPGAs. It offers flexibility to tradeoff between fault tolerance and hardware cost by applying TMR only to critical channels of the NN layers, previously identified by means of error injection. The fault-tolerant accelerators automatically generated preserve their throughput and incur in very low latency increase. Additionally, experiments show how FAT in conjunction with STMR provides higher benefits than SAT-trained NNs.

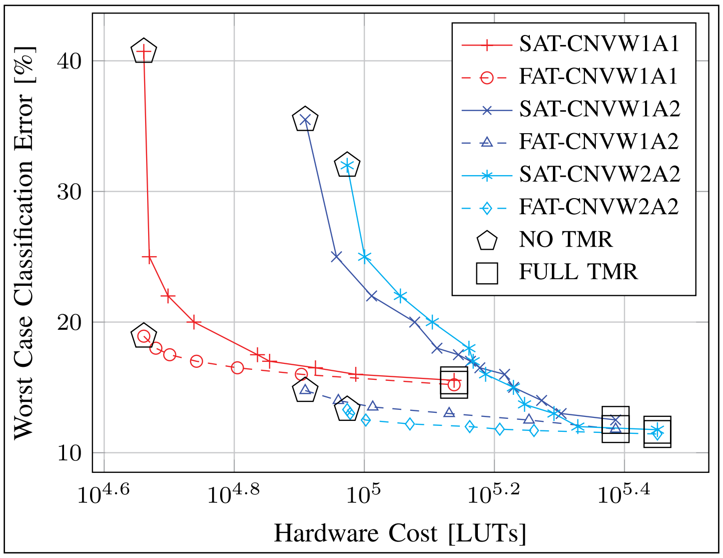


Figure 3. Design space of worst-case error versus hardware cost for CNVW1A1, CNVW1A2, and CNVW2A2 with the throughput of ≈21 k frames/s.

For future work, we plan to: validate the STMR approach under high-radiation environments; develop robust training methods that are specific to STMR, in particular, methods that extend FAT to support a small number of outliers; and finally, extend the STMR tool flow to support more NN layer types, such as long short-term memories (LSTMs). ■

References

[1] U. Zahid et al., “FAT: Training neural networks for reliable inference under hardware faults,” in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2020, pp. 1–10.

[2] G. Gambardella et al., “Efficient error-tolerant quantized neural network accelerators,” in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Oct. 2019, pp. 1–6.

[3] F. Libano et al., “How reduced data precision and degree of parallelism impact the reliability of convolutional neural networks on FPGAs,” *IEEE Trans. Nucl. Sci.*, vol. 68, no. 5, pp. 865–872, May 2021.

[4] D. Xu et al., “Reliability evaluation and analysis of FPGA-based neural network acceleration system,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 3, pp. 472–484, Mar. 2021.

[5] F. Brosser et al., “Assessing scrubbing techniques for Xilinx SRAM-based FPGAs in space applications,” in *Proc. Int. Conf. Field-Program. Technol. (FPT)*, Dec. 2014, pp. 296–299.

[6] H.-B. Wang et al., “Impact of single-event upsets on convolutional neural networks in Xilinx Zynq FPGAs,” *IEEE Trans. Nucl. Sci.*, vol. 68, no. 4, pp. 394–401, Apr. 2021.

[7] F. Libano et al., “Selective hardening for neural networks in FPGAs,” *IEEE Trans. Nucl. Sci.*, vol. 66, no. 1, pp. 216–222, Jan. 2019.

[8] N. Khoshavi et al., “SHIELDeNN: Online accelerated framework for fault-tolerant deep neural network architectures,” in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2020, pp. 1–6.

[9] T. Spyrou et al., “Neuron fault tolerance in spiking neural networks,” in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Feb. 2021, pp. 743–748.

[10] K. Zhao et al., “FT-CNN: Algorithm-based fault tolerance for convolutional neural networks,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 7, pp. 1677–1689, Jul. 2021.

[11] Y. Umuroglu et al., “FINN: A framework for fast, scalable binarized neural network inference,” in *Proc. FPGA*, 2016, pp. 65–74.

[12] I. Hubara et al., “Binarized neural networks,” in *Advances in Neural Information Processing Systems*, vol. 29, D. Lee et al., Eds. Red Hook, NY, USA: Curran Associates, 2016.

[13] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” M.S. thesis, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, 2009.

■ Direct questions and comments about this article to Giulio Gambardella, Xilinx Research Labs, Dublin, Ireland; giuliog@xilinx.com.

API-Based Hardware Fault Simulation for DNN Accelerators

Patrik Omland, Yang Peng, and Michael Paulitsch
Dependability Research Laboratory
Intel Deutschland GmbH
85579 Neubiberg, Germany

Jorge Parra, Gustavo Espinosa, and Abishai Daniel
Intel Corporation, Santa Clara, CA 95054 USA

Gereon Hinz
STTech
82031 Grünwald, Germany

Alois Knoll
Department of Informatics
Technical University Munich
85748 Munich, Germany

Editor's notes:
This article presents an application program interface (API)-based hardware fault simulation method to investigate the effect of hardware faults on the failure probability of deep neural network (DNN) accelerators.
—*Fei Su, Intel Corporation*

CONTINUED TRANSISTOR SCALING results in lower operating voltages that enable increased levels of integration within a given silicon area footprint. However, it also entails an increase in the likelihood of unintended bit flips and data corruption at the device level.

The rate of these faults per computational resource requires special consideration when

- combining many computational resources (e.g., supercomputers and server farms) and
- executing applications with high dependability requirements, such as in automotive (requiring failure rates below 10^{-8} failures/hour for safety-critical functions).

To reduce the likelihood of data corruption, hardware designers identify high-risk components and add protection circuitry, such as parity checks and error correction codes (ECCs). However, protection circuitry requires die area and increases power consumption which could otherwise be used to increase performance. The more comprehensive the protection, the higher the error detection or correction capabilities, but the more area it occupies.

A balance must be found in the tradeoff between an integrated circuit's dependability and performance. Experiments indicate that deep neural networks (DNNs) are more resilient to hardware faults than other programs.[1] In this context, special "DNN accelerators" have been designed for efficient DNN execution [3]. These may require lower than usual levels of hardware protection while satisfying the same dependability targets for DNN applications.

So what is the probability of output failure due to hardware faults for DNNs running on these DNN accelerators? In this work, we present a novel method for estimating this probability. Our approach works by expanding the primitives of application program interfaces (APIs) used by DNNs with hardware-specific fault simulations: First, the original primitive is run, then the output is modified in the way it would be corrupted due to faults in the target hardware. The actual hardware is not required. By executing a DNN with this modified API simulating hardware faults, statistics may be generated on output failures. Unlike existing approaches, our approach uniquely combines.

Digital Object Identifier 10.1109/MDAT.2022.3180977
Date of publication: 8 June 2022 ; date of current version: 10 March 2023.

[1]Compare bit error rate thresholds found in [1] with requirements in [2].

- *Accuracy*: The actual workload is run on an accurate hardware fault simulation.
- *Speed*: The simulation time is not constrained by the lack of nor the speed of hardware to be simulated.
- *Scale*: By sharing the modified API implementation, accurate dependability estimates for specific workloads may be generated without hardware/algorithmic knowledge.

DNN accelerators

Computing platforms tailored specifically to the needs of DNNs have become more common over the past years. Prominent examples are Google Tensor Processing Units, Nvidia Tensor Cores, Intel Xeon Tile Matrix Multiply Units, and Intel Xe HPC graphics processing units (GPUs) [3].

By far, most computer operations carried out by DNNs are spent on matrix multiplication: In DNN terminology, the fully connected and convolution layers are calculated by algorithms using matrix multiplication.[2] For DNNs such as ResNet-50, these multiplications involve large matrices with dimensions, n, in the thousands. Matrix multiplication is, approximately, an $O(n^3)$ operation. All other commonly used DNN operations are $O(n)$ operations. Consequently, accelerators geared toward DNNs specifically aim to accelerate large matrix multiplications. Most of them

- adopt an architecture that consists of systolic arrays (SAs) operating in parallel and
- feature a memory hierarchy designed to maximize the reuse of data cached close to the SAs,

[2]Chetlur *et al.* [4] explain how to convert convolution to matrix multiplication.

where each SA computes small matrix-multiply-accumulate (MMA) operations, $D = A \cdot B + C$.

We will refer to this class of accelerators as "DNN accelerators." The typical architecture of a DNN accelerator is shown in Figure 1. The white blocks inside the SA represent multiply-accumulate-fused (MAF) units, performing the actual calculations.

When designing DNN accelerators, the relative robustness of DNNs with reference to hardware faults is taken advantage of by optimizing the level of hardware protection for performance gains. In this context, SAs and their caches present particularly good opportunities for such gains.

Protection circuitry for large caches (L4–L2 in Figure 1) requires relatively little die area. In comparison, the SA caches (L1 in Figure 1) are very small and there may be thousands of them—here, protection carries a high-performance cost. Analogously, while an ALU on the "slice level" in Figure 1 may be implemented with hardware protection, doing the same for each of the dozens of MAF units comprising a single SA places a large burden on performance.

Related work

Many methods of estimating the likelihood of program failure due to hardware faults exist. Below, we present the most prominent ones.

Statistical fault injection

In statistical fault injection, faults are injected at program runtime. These faults may be injected at different system abstraction levels (gate, microarchitecture, and so on). In general, lower-level fault injection provides more accurate results but may not be scalable in practice due to long execution times, while higher-level fault injection may run much faster, but at the price of lower accuracy [5].

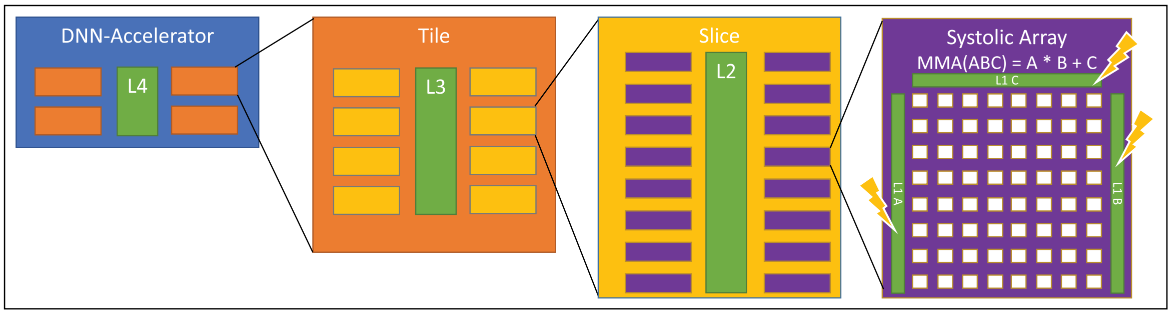


Figure 1. Typical architecture of a DNN accelerator. Memory hierarchy depth (L4–L1) and the number of units on each level (4, 8, 16) chosen arbitrarily.

Hierarchical simulations have been applied to address this tradeoff by simulating different parts of the system at different abstraction levels so that required details are modeled only for the parts of interest [5]. The proposed method in this work follows a similar concept as hierarchical simulations.

Vulnerability factors

In the vulnerability factor approach, simulating lower system abstraction levels individually for each program is avoided by estimating the fraction of faults affecting a given level from the next lower level. Frequently used factors are the hardware vulnerability factor (HVF) [6], the program vulnerability factor (PVF) [7], and the timing vulnerability factor (TVF) [8]. The overall failure rate for a program, P, is then estimated by (1), where F denotes the fraction of time in a particular use condition, uc, itself dependent on the clock frequency, f_{clk}

$$\text{Failure Rate}(P) \approx \sum_{uc} F_{uc,P}(f_{\mathrm{clk}}) \cdot \sum_{c \in \text{circuits}} \text{Fault-Rate}_c \cdot \mathrm{TVF}_{uc,c} \cdot \mathrm{HVF}_{uc,c} \cdot \mathrm{PVF}_{uc,c,P}. \quad (1)$$

However, not much is gained if $\mathrm{PVF}_{uc,c,P}$ has to be estimated individually for each DNN, each use condition, and each circuit.[3] As will be shown in the upcoming section, hierarchical fault injection simulations not only deliver more accuracy, but may be implemented in a general, scalable fashion.

Evaluating vulnerability of DNN-based applications

To understand the vulnerability of DNN-based applications, many existing works (e.g., [1] and [9]) adopt application-level fault injection by, say, injecting faults directly into the DNN model (e.g., weights). However, this approach does not reflect the actual impact of the underlying platform on which the DNN is executed. As will be shown in the upcoming section, microarchitectural details of DNN accelerator designs have a profound impact on how hardware faults propagate to the level of the DNN model.

Problem statement

The task at hand is a risk assessment for DNNs when facing hardware faults on DNN accelerators. Generally, given a program, p, the risk of a hardware fault, f, causing failure with severity $\in \{0 = \text{none}, 1, \ldots\}$, may be defined as

$$\text{risk} = \Pr(f, \text{severity} \mid p) \cdot \text{severity} \quad (2)$$

commonly known as the risk matrix approach, where shorthand Pr denotes probability.

The probability on the right-hand side of (2) may be separated into two parts

$$\Pr(f, \text{severity} \mid p) = \underbrace{\Pr(f \mid p)}_{\text{exposure}} \cdot \underbrace{\Pr(\text{severity} \mid f, p)}_{\text{conditional failure}} . \quad (3)$$

The "exposure probability" measures the likelihood of the fault, f, occurring while a given program, p, is exposed to it. For instance, if a program makes no use of floats, and the hardware fault considered is a fault in an floating-point unit (FPU), the program's exposure probability to that fault equals zero.

The "conditional failure probability" measures the likelihood of the program, p, failing with severity, conditional on it being exposed to a fault, f. For instance, if the program's output is a single-precision floating-point value and the fault only ever flips the least significant bit of that value, the relative output error equals 2^{-23}: For most programs, this error will not be considered program failure, so the associated conditional failure probability would equal zero.

As calculating the risk using (2) becomes trivial once the failure probability (3) has been estimated, moving forward, we only consider the latter problem.

Novel API-based fault simulation

Numerical programs, in particular, DNNs, rely on standards-based APIs to implement mathematical operations such as matrix multiplication. The actual operation is usually implemented by the hardware manufacturer, requiring intimate knowledge of the accelerator's memory hierarchy, instruction pipelining, and so on. In the proposed approach, hardware fault simulations are implemented into these APIs for the very same reason. Also, fault simulations thus implemented become available immediately to every program linking the given API.

[3]For many central processing unit (CPU) applications, the approximation $\mathrm{PVF}_{uc,c,p} \approx 1$ may be used, making this approach useful for rough estimates. However, in this work, we are particularly interested in the $\mathrm{PVF}_{uc,c,p} \ll 1$ property of DNNs.

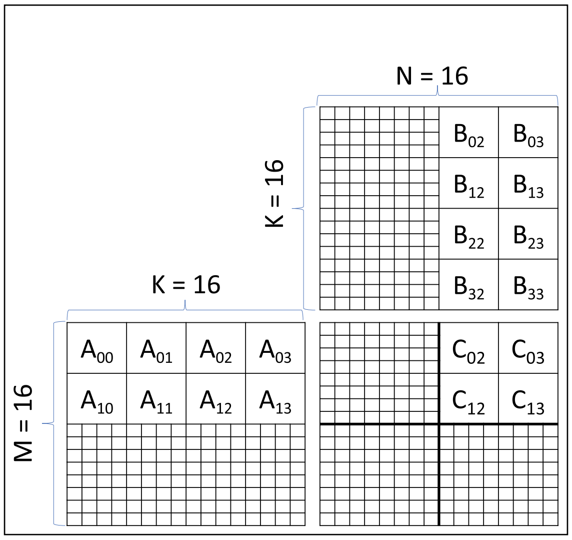


Figure 2. Matrix multiplication on DNN accelerators.

The proposed API-based fault simulation for a given API comprises the following steps: For each API operation executed on the accelerator

1) *Model of computation (MoC):* Develop an MoC, modeling how the operation is executed on the actual hardware.
2) *Fault MoC-scope:* For the hardware fault under consideration, find the execution steps affected in the MoC by one such fault.
3) *API fault simulation*: Develop a fault simulation for these execution steps making as much use of the API operation's (efficiently computed) output as possible and including the simulation with the API operation.

Without loss of generality, we provide a sample application with a simplified MoC, following the steps outlined above, to illustrate the proposed method.

Simplified model of computation

We implement the fictitious general matrix multiply API function GEMM16 $(A, B) = A^{16\times16} \cdot B^{16\times16}$, on a DNN accelerator featuring four SAs. Each SA itself may execute an MMA instruction, $A^{4\times4} \cdot B^{4\times4} + C^{4\times4}$. The generalization to arbitrary dimensions and the number of SAs is straightforward.

The multiplication is depicted in Figure 2. The 16 submatrices $C_{mn}{}^{4\times4}$ may be calculated by

$$C_{mn}^{4\times4} = \sum_{k=0}^{k<4} A_{mk}^{4\times4} B_{kn}^{4\times4}. \quad (4)$$

Algorithm 1. GEMM16: returns $A^{16\times16} \cdot B^{16\times16}$ using four SAs capable of $A^{4\times4} \cdot B^{4\times4} + C^{4\times4}$-MMA

```
Input: Matrices $A^{16\times16}$, $B^{16\times16}$
Output: $C = A \cdot B$
1: $C = 0$
2: for $SA = 0$ to $3$ do
3:     $m_0 = \lfloor SA/2 \rfloor * 2$
4:     $n_0 = \lfloor SA \mod 2 \rfloor * 2$
5:     for $k = 0$ to $3$ do
6:         for $m = m_0$ to $m_0 + 1$ do
7:             for $n = n_0$ to $n_0 + 1$ do
8:                 $C_{mn} = \mathrm{MMA}_{SA}(A_{mk}, B_{kn}, C_{mn})$
9: return $C$
```

The GEMM16 algorithm using MMA instructions is given by Algorithm 1. It divides *C* into quadrants, each assigned one SA (see Figure 2).

Unrolling the *m*, *n* loops for the upper right quadrant we get

```
1 for $k = 0$ to $3$ do
2     $C_{02} = \mathrm{MMA}_1(A_{0k}, B_{k2}, C_{02})$
3     $C_{03} = \mathrm{MMA}_1(A_{0k}, B_{k3}, C_{03})$
4     $C_{12} = \mathrm{MMA}_1(A_{1k}, B_{k2}, C_{12})$
5     $C_{13} = \mathrm{MMA}_1(A_{1k}, B_{k3}, C_{13})$.
```

Notice that each *k*-iteration requires only four different *A* and *B* inputs, namely A_{0k}, A_{1k}, B_{k2}, and B_{k3}. Now, consider the memory hierarchy in Figure 1: For L1A (L1B) large enough to cache one (two) 4×4-submatrices, data requests to L2 for these inputs are halved.[4] Moving forward we assume just that.

Simulating transient L1 cache faults

Suppose one of the L1 caches of the upper right quadrant's SA experiences a transient bit-flip—what is the fault's MoC-scope? From the unrolled loop above, we see that any such fault is confined to one *k*-iteration (data is not reused across *k*-iterations) and affects at most two C_{mn} (e.g., if B_{12} is corrupted in line 2, it affects C_{02} and then C_{12} in line 4).

Next, we develop the API fault simulation. In the unrolled loop above, suppose the fault occurs at iteration $k = 2$: line 4 and has the effect $B_{23} \overset{\text{↯}}{\mapsto} \tilde{B}_{23}$. The corresponding effect on the output, $C \overset{\text{↯}}{\mapsto} \tilde{C}$, reads

$$\tilde{C}_{12} = C_{12} - \mathrm{MMA}_1(A_{12}, B_{23}, 0) + \mathrm{MMA}_1(A_{12}, \tilde{B}_{23}, 0).$$

As C_{12} is returned by the regular API operation, we do not need to calculate it ourselves but can utilize

[4]For real-world DNN accelerators with $M \times K \times N$-MMA: If L1A caches a single $M \times K$-submatrix and L1B caches L_B $K \times N$-submatrices, each SA may be assigned $L_B \times L_B$ $M \times N$-submatrices in the output to reduce L2-requests for *A*, *B* by a factor of $1/L_B$ using Algorithm 1.

Algorithm 2. GEMM16_FI_C: Simulate L1A/B/C cache fault during GEMM16 execution.

```
Input: $A$, $B$, $C = \text{GEMM16}(A, B)$, $SA$, $(k, m, n)$, $X$
Output: $\tilde{C}$, fault-simulated output of GEMM16
1: $m = m + \lfloor SA/2 \rfloor * 2$
2: $n = n + \lfloor SA \mod 2 \rfloor * 2$
3: if $(X == A)$ then
4:     if $(n == 0)$ then
5:         $C_{m0} = \text{MMA}(-A_{mk}, B_{k0}, C_{m0})$
6:         $C_{m0} = \text{MMA}(\tilde{A}_{mk}, B_{k0}, C_{m0})$
7:     $C_{m1} = \text{MMA}(-A_{mk}, B_{k1}, C_{m1})$
8:     $C_{m1} = \text{MMA}(\tilde{A}_{mk}, B_{k1}, C_{m1})$
9: else if $(X == B_0)$ then
10:     ...
11: ...
12: return $C$
```

the high-performance API implementation as input to the simulation.

More generally, the effect of a cache fault occurring in SA, at iteration $(k, m, n) \in [0, 3] \times [0, 1] \times [0, 1]$ and cache index $X \in \{A, B_0, B_1, C\}$, may be modeled by Algorithm 2.[5]

Note that in Algorithm 2, timing matters: If a fault in L1A happens at $n = 0$, then two of C's 4×4-submatrices are affected, otherwise only one. Similarly, if L1B suffers a fault corrupting B_0 at $n = m = 1$, C will not be affected.

The API hardware fault simulation is listed in Algorithm 3. To inject one random fault into a program making multiple uses of GEMM16, we count the overall MMA instruction calls, *MMA_total*, of that program, and pick a positive random number, *MMA_FI* ≤ *MMA_total*, representing one of these calls.

By far, most of the work in Algorithm 3 is performed through the API call to GEMM16: This will be executed with maximal performance on any hardware with a GEMM16 implementation. In comparison, the up to two MMA calls from GEMM16_FI are insignificant—in particular, for real-world large GEMM operations with thousands of MMA calls.

Coming back to the original problem of estimating (3): We may approximate Pr(severity | f, p) by the relative failure frequency of program runs with hardware fault simulation. To estimate Pr(f | p), the likelihood of encountering a transient fault, random in time and space, does not depend on the level of parallelization: Whether four SAs are used, or a single one four-times as long, does not matter. Accordingly,

Algorithm 3. GEMM16_FSIM: simulate fault in GEMM16 execution.

```
Input: $A$, $B$. Global: $MMA\_Calls$, $MMA\_FI$
Output: $\tilde{C}$, fault-simulated output of GEMM16
1: $C = \text{GEMM16}(A, B)$
2: if $MMA\_FI \in [MMA\_Calls, MMA\_Calls + 64)$ then
3:     Choose random $(SA, k, m, n, X)$
4:     $C = \text{GEMM16\_FI\_C}(A, B, C, SA, k, m, n, X)$
5: $MMA\_Calls = MMA\_Calls + 64$
6: return $C$
```

Algorithm 4. GEMM16_FI_L: simulate fault inside SA logic during GEMM16 execution.

```
Input: $A$, $B$, $C = \text{GEMM16}(A, B)$, $SA$, $(k_{FI}, m, n)$
Output: $\tilde{C}$, fault-simulated output of GEMM16
1: $m = m + \lfloor SA/2 \rfloor * 2$
2: $n = n + \lfloor SA \mod 2 \rfloor * 2$
3: $C_{mn} = 0$
4: for $k = 0$ to $3$ do
5:     if $k \neq k_{FI}$ then
6:         $C_{mn} = \text{MMA}_{SA}(A_{mk}, B_{kn}, C_{mn})$
7:     else
8:         $C_{mn} = \widetilde{\text{MMA}}_{SA}(A_{mk}, B_{kn}, C_{mn})$
9: return $C$
```

given the fault rate, R_f, of one L1 cache and the duration, τ_{MMA}, of one MMA execution, we may estimate

$$\Pr(f \mid p) \approx 1 - \exp(-\text{MMA\_total} \cdot \tau_{\text{MMA}} \cdot R_f) \quad (5)$$

where the exponential failure distribution was used to model the probability of fault given fault rate and duration.

Simulating transient faults inside SAs

The same method applied for simulating transient faults in the SA's caches (previous section) may be used for the simulation of arbitrary faults inside the SAs $\text{MMA} \xmapsto{f} \widetilde{\text{MMA}}$. For the SA's digital arithmetic, however, simulating the correct C_{mn}-input to the MMA instruction matters[6] and thus needs to be calculated by simulating the k-loop (see Algorithm 4). For real-world applications with large k-loops, the additional simulation overhead is notable.

Simulating permanent faults

A permanent fault in an SA or its caches affects every k, m, and n and thus Algorithm 4 needs to be modified accordingly. The challenge in simulating permanent faults lies in modeling the likelihood of encountering the faulty SA.

[5]By not using the actual C_{mn}-input to MMA for the given (k, m, n), Algorithm 2 does not account for "$(a + b) + c \neq a + (b + c)$." To account for that, the k-loop needs to be executed as in Algorithm 4.

[6]The SA may, for instance, perform optimizations if $C_{mn} = 0$.

Suppose we execute a program with ten GEMM16 invocations on a DNN accelerator with 16 SAs. As GEMM16 requires four SAs, each time GEMM16 is invoked, so there are 16!/(16–4)! = 43,680 ways of assigning four output quadrants to 16 SAs. For the whole program, we get $43{,}680^{10} \approx 10^{46}$ possibilities.

One approach to handle this problem is to model worst- and best-case scenarios. For instance, considering Figure 1, in the worst case, the program's GEMM16s might always be mapped to the same slice and randomly distributed among its 16 SAs, one of which has a permanent fault. In the best case, each SA might be chosen at random from the 512 SAs comprising Figure l's DNN accelerator.

ResNet-50 proof of concept

We applied the method presented in the previous section to ResNet-50 [10] inference on several DNN accelerator configurations by modifying the oneDNN API. Two of the oneDNN operations used by ResNet-50 utilize SAs: Matrix multiplication and convolution.[2] We analyzed the algorithms implemented by oneDNN for DNN accelerators and developed fault models according to the method described above. The buffers (FP16 data format) were corrupted by a single random transient bitflip for each inference, analogous to Algorithm 2. 24.8k ImageNet [11] inferences were executed for each configuration. The results are shown in Table 1.

The simulation was run on an Intel i9-7960X CPU. A single inference without fault injection took 104 ms. The overhead in Table 1 is given with reference to this duration. "$M \times K \times$ N" specifies the MMA dimensions and "L_B " the number of $K \times N$ matrices cached in L1B.[4] "ΔTop" lists the change in percentage of inputs for which the highest rated output label is correct with versus without fault simulation. "#MMA" lists the number of MMA calls for a single inference.

Table 1. Transient buffer fault simulation for DNN accelerators running ResNet-50 inference.

$M \times K \times N$	L_B	#MMA	ΔTop [%]	Overh.[%]
$32 \times 32 \times 32$	4	117,216	-1.12	11.0
	2		-1.05	3.3
$16 \times 16 \times 16$	4	994,368	-1.07	1.2
	2		-0.92	0.5
$8 \times 8 \times 8$	4	7,923,456	-0.96	0.1
	2		-0.76	0.0

As expected from the previous section, the conditional failure probability (3), which may be identified with "ΔTop," decreased with decreasing MMA dimensions. The corrupted buffer element affects fewer output elements. The effect on the exposure probability (3) is more complicated: While smaller buffers result in a smaller frequency of buffer corruption, accounting for the time the application is exposed to these buffers is not straightforward. While the number of required MMA calls obviously increases with decreasing MMA dimensions, estimating the duration of each such call for different dimensions requires knowledge of the SA's implementation. Consequently, one should not draw conclusions on the risk (2) associated with different SA configurations from Table 1 without accounting for these factors.

In conclusion, we successfully applied our novel methodology to a large workload, performing hundreds of thousand hardware fault simulations within hours on a regular CPU, where more traditional approaches would have taken days for a single simulation.

Future work

The best- and worst-case approaches for modeling permanent faults (previous section) do not yield the single probability for program failure we are after (3). Rather, it delivers upper/lower bounds on that probability. Moving forward, we are developing models of computation incorporating scheduling algorithms for DNN accelerators to accurately estimate this probability.

In the previous section, we suggest running a hardware simulation, $\widetilde{\text{MMA}}$, for the complete MMA instruction. When modeling permanent faults inside the SAs, this simulation overhead becomes significant. In future work, we will develop methods reducing the simulation overhead to simulating single MAF units (previous section) only.

FINALLY, WHILE OUR research has focused on utilizing DNN accelerators for the class of DNN programs, other classes of matrix multiplication heavy programs would profit from using DNN accelerators (e.g., finite-element methods). In upcoming work, we will investigate the effect of hardware protection design choices on the dependability of these kinds of programs. ■

Acknowledgments

We would like to thank Yue Qi, Fangwen Fu, and Mourad Gouicem for technical insights on DNN accelerators and algorithms utilizing their architecture.

References

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

[2] H. T. Nguyen et al., "Chip-level soft error estimation method," *IEEE Trans. Device Mater. Rel.*, vol. 5, no. 3, pp. 365–381, Sep. 2005.

[3] A. Rodriguez, *Deep Learning Systems: Algorithms, Compilers, and Processors for Large-Scale Production*. San Rafael, CA, USA: Morgan & Claypool, 2020.

[4] S. Chetlur et al., "CuDNN: Efficient primitives for deep learning," Oct. 2014, *arXiv:1410.0759*.

[5] Z. Kalbarczyk et al., "Hierarchical simulation approach to accurate fault modeling for system dependability evaluation," *IEEE Trans. Softw. Eng.*, vol. 25, no. 5, pp. 619–632, Oct. 1999.

[6] V. Sridharan and D. R. Kaeli, "Using hardware vulnerability factors to enhance AVF analysis," in *Proc. Int. Symp. Comput. Archit. (ISCA)*, 2010, pp. 461–472.

[7] V. Sridharan and D. R. Kaeli, "Eliminating microarchitectural dependency from architectural vulnerability," in *Proc. Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2009, pp. 117–128.

[8] N. Seifert and N. Tam, "Timing vulnerability factors of sequentials," *IEEE Trans. Device Mater. Rel.*, vol. 4, no. 3, pp. 516–522, Sep. 2004.

[9] G. Li et al., "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal. (ACM)*, 2017, pp. 1–12.

[10] K. He et al., "Deep residual learning for image recognition," 2015, *arXiv:1512.03385*.

[11] J. Deng et al., "ImageNet: A large-scale hierarchical image database," in *Proc. CVPR*, 2009, pp. 248–255.

Patrik Omland is a research scientist with Intel Deutschland GmbH, 85579 Neubiberg, Germany. He is pursuing a PhD with the Department of Informatics, Technical University Munich, Munich, Germany. His research interests include the effect of hardware faults on program execution and digital arithmetic/numerical algorithms. Omland has a master's in mathematical physics from the Ludwig-Maximilians-University Munich, Munich.

Yang Peng is a research scientist and system architect with Intel Deutschland GmbH, 85579 Neubiberg, Germany. His research interest includes system architecture for dependable artificial intelligence/machine learning (AI/ML)-based systems. Peng has a PhD in electrical engineering from the Technical University of Munich, Munich, Germany.

Michael Paulitsch is a principal engineer with Intel Deutschland GmbH, 85579 Neubiberg, Germany, where he leads the Dependability Research Laboratories. His research interests include novel architectures for dependable systems and machine learning. Paulitsch has a PhD from Technical University Vienna, Vienna, Austria, and a PhD from the Vienna University of Economics and Business, Vienna. He is a Senior Member of IEEE.

Jorge Parra is a computer architect with Intel Corporation, Santa Clara, CA 95054 USA, working on Intel's Xe GPU products. His research interests include computer architecture, machine learning hardware architectures, and artificial intelligence. Parra has a PhD and an MSc in electrical engineering from the University of New Mexico, Albuquerque, NM, USA.

Gustavo Espinosa is a senior principal engineer with Intel Corporation, Santa Clara, CA 95054 USA, where he leads reliability and security architecture development for discrete GPU products. Espinosa has a master's in computer engineering from Boston University, Boston, MA, USA. He is a member of IEEE.

Abishai Daniel is a staff reliability, availability and serviceability (RAS) quality and reliability engineer with Intel Corporation, Santa Clara, CA 95054 USA, with a focus on statistical predictive model development and application of novel machine learning techniques to reliability modeling. Abishai has an MSEE and a PhD from the University of Michigan, Ann Arbor, MI, USA.

Gereon Hinz is the CEO of STTech, 82031 Grünwald, Germany, providing solutions to current and upcoming technological challenges in the autonomous systems domain. Hinz has a master's in cybernetics from the University of Stuttgart, Stuttgart, Germany.

Alois Knoll is a professor of computer science with the Department of Informatics, Technical University Munich, 85748 Munich, Germany. His research interests include robotics, artificial intelligence, and realtime systems. He is a Senior Member of IEEE.

■ Direct questions and comments about this article to Patrik Omland, Dependability Research Laboratory, Intel Deutschland GmbH, 85579 Neubiberg, Germany; patrik.omland@intel.com.

On the Impact of Uncertainties in Silicon-Photonic Neural Networks

Sanmitra Banerjee
Department of Electrical and Computer Engineering
Duke University
Durham, NC 27708 USA

Krishnendu Chakrabarty
Department of Electrical and Computer Engineering
Duke University
Durham, NC 27708 USA

Mahdi Nikdast
Department of Electrical and Computer Engineering
Colorado State University
Fort Collins, CO 80523, USA

Editor's notes:
This article presents a method of criticality assessment to identify susceptible components of silicon-photonic neural networks.
—Fei Su, Intel Corporation

The rise of deep learning as the foundation of most modern artificial intelligence (AI) applications has been fueled by domain-specific AI accelerators that support custom memory hierarchies, variable precision, and optimized matrix multiplication. Modern AI accelerators demonstrate superior energy and footprint efficiency compared to GPUs for a variety of inference and some training tasks. With the slowdown of Moore's law, these accelerators approach fundamental limits on their performance due to: 1) the limited computational and performance-per-watt capabilities of silicon CMOS and 2) the use of low-bandwidth metallic interconnects [1].

Digital Object Identifier 10.1109/MDAT.2022.3161599
Date of publication: 2 May 2022; date of current version: 10 March 2023.

Optical computing and communication can potentially overcome both these performance-limiting issues. Computations required in deep learning, such as matrix–vector multiplication, can be performed entirely in the optical domain with high energy efficiency. For instance, with respect to multiply and accumulate (MAC) operations, optical computing can achieve a 1,000× better energy footprint efficiency compared to the most energy-efficient electronic accelerators today [2]. Additionally, optical interconnects represent a post-Moore's law alternative to replace low-performance metallic interconnects, hence ensuring lower power consumption, higher bandwidth, and lower latency for the communication.

With the advent of silicon photonics, optical components can now be integrated into dense silicon chips using CMOS-compatible manufacturing techniques. Silicon-photonic neural networks (SPNNs) integrate the performance benefits offered by optical computing and interconnects with the low-cost and mature CMOS fabrication process to enable low-latency and energy-efficient optical domain data transport and processing. However, SPNNs are prone to several reliability issues. Imperfections in the optical lithography process lead to variations in critical waveguide dimensions and hence incorrect operation of photonic components. Moreover, mutual thermal crosstalk between adjacent optical components due to convective heat transfer has been observed [3]. These uncertainties, along with the finite encoding precision on tuning parameters, can lead to the erroneous matrix–vector multiplication and a consequent loss in SPNN classification accuracy.

In this article, we present a comprehensive analysis of the impact of uncertainties in SPNNs. In particular, we show that the effect of uncertainties can vary depending on the location and type of affected optical components. The main contributions of this article are as follows.

- An overview of different uncertainties in SPNNs originating from fabrication-process variations, manufacturing defects, and thermal crosstalk.
- A hierarchical analysis of the impact of different uncertainties on SPNN performance starting from the component level to the system level.
- A framework to identify critical SPNN components where uncertainties can lead to severe performance degradation.

Overview of SPNNs

A multilayer perceptron-based artificial neural network (ANN) maps an input feature vector to an output vector through a series of linear transformations and nonlinear activation functions. The neurons in adjacent linear layers (Figure 1a) are interconnected using weighted edges; these weights are updated during training to change the effect of each input. To mimic this dynamic weighting of connections, silicon-photonic devices can be used to control the optical transmission between two neurons in different ways. Coherent SPNNs (C-SPNNs) use thermo-optic phase shifters (PhSs) to modify the phase of the optical signal between two neurons. In this case, the tuned phase shifts in the PhS denote the dynamic edge weight. Alternatively, noncoherent SPNNs (N-SPNNs) use microring resonators (MRs) to modify the optical signal power on the interconnection between two neurons. The performance of NSPNNs can be adversely affected due to geometric variations in the waveguides. Experimental studies have shown that MRs used in N-SPNNs can suffer from a 4.79-nm resonance drift within a wafer due to process variations [4]. Additionally, NSPNNs require several power-hungry wavelength-conversion steps and are prone to interchannel crosstalk among different wavelengths.

As a result, C-SPNNs are being preferred for emerging AI accelerators [5]. In this article, we primarily focus on uncertainties in C-SPNNs and present an overview of uncertainties in N-SPNNs in the upcoming section. Fully connected layers in C-SPNNs can be represented mathematically as matrix–vector multiplication followed by an activation function. Consider a layer L_i with n_i neurons fully connected to the next layer L_{i+1} with n_{i+1} neurons. The output vector at L_{i+1} is then given by $O_{i+1}^{n_{i+1}\times 1} = f_{i+1}\left(M_{i+1}^{n_{i+1}\times n_i} O_i^{n_i\times 1}\right)$. Note that f_{i+1} and M_{i+1} are the nonlinear activation function and weight matrix associated with layer L_{i+1}, respectively. In C-SPNNs, the linear multiplication with the weight matrix (i.e., M) is implemented using arrays of configurable Mach–Zehnder interferometers (MZIs), as shown in Figure 1b. Typically, activation functions (e.g., f_{i+1}) are implemented electronically, as optical

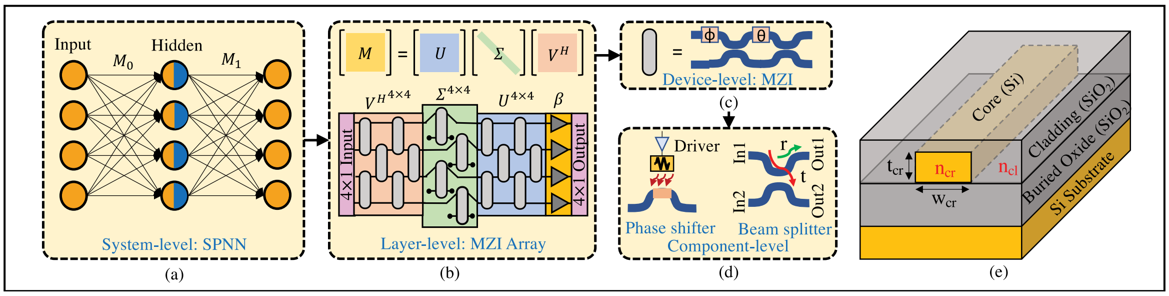


Figure 1. Hierarchical design of a C-SPNN. (a) System level. (b) Layer level. (c) Device level. (d) Component level. (e) 3-D schematic of a strip waveguide.

nonlinearities require high signal power and impose lower bounds on the physical footprint [6].

MZIs are used to determine the phase difference between collimated optical signals. Figure 1c shows the typical structure of an MZI with two tunable PhSs—with phase shifts ϕ and θ—and two 50:50 beam splitters (BeSs). The PhS, as shown in Figure 1d, are used to apply phase shifts and obtain varying degrees of interference between the optical signals traversing the two waveguides in the MZI. The refractive index of a silicon (Si) waveguide changes with temperature; this is known as the thermo-optic effect. The thermal microheaters in PhSs can tune this temperature change by varying the current through a resistor coil. The Joule heat dissipated from the resistor, in turn, controls the applied phase shift. Figure 1d also shows the schematic of a 2 × 2 directional coupler-based BeS. A fraction of the input optical signal denoted by transmittance t in In1 (In2) is coupled to Out2 (Out1) with a phase shift of $\pi/2$. The remaining fraction of the optical signal denoted by reflectance r is reflected to the original waveguide and propagates from In1 (In2) to Out1 (Out2). The ratios r and t are referred to as splitting ratios in this article. As the optical signal is distributed among the two waveguides in the ratios of r and t, the optical power is distributed in the ratios of r^2 and t^2. Therefore, from the law of conservation of energy, we have $r^2 + t^2 = 1$. In an ideal 50:50 BeS, half of the optical power is reflected, while the other half is transmitted; therefore, both the transmittance and reflectance coefficients are $\frac{1}{\sqrt{2}}$. The transfer matrix of an MZI with two PhSs (ϕ and θ) and two BeSs—with splitting ratios (r, t) and (r', t')—is given by

$$T_{\text{MZI}} = \begin{pmatrix} rr'e^{i(\theta+\phi)} - tt'e^{i\phi} & ir'te^{i\theta} + it'r \\ it'r e^{i(\theta+\phi)} + itr'e^{i\phi} & -tt'e^{i\theta} + rr' \end{pmatrix}. \quad (1)$$

Using singular value decomposition, the weight matrix corresponding to layer L_i can be factorized into two unitary matrices and a diagonal matrix: $M_i = U_i \Sigma_i V_i^H; U_i$ and V_i are the unitary matrices and V_i^{H} denotes the Hermitian transpose of V_i. Moreover, Σ_i is a diagonal matrix consisting of the eigenvalues of M_i. Any $n_i \times n_i$ unitary matrix can be represented by an array of $\binom{n_i}{2}$ MZIs connected, as shown in Figure 1b. MZIs can also be used to attenuate each waveguide separately without mixing (see $\Sigma^{4\times4}$ in Figure 1b). In this way, an $n_i \times n_i$ diagonal matrix can be represented by n_i MZIs with one input and one output of each MZI terminated using optical waveguide tapers to prevent back-reflection and cross-coupling at the unused ports [7]. Additionally, an optical amplification, denoted by β in Figure 1b, is required on each output to counter the power dissipation in lossy MZIs.

SPNNs can be trained either in an *in-situ* or an *ex-situ* fashion. In *in-situ* training, gradient computation needs to be performed on the SPNN platform; this involves sequentially perturbing each parameter of the circuit. Such training demands significant computational time and resources and its efficiency can be affected under thermal crosstalk. Thus, current implementations of SPNNs are typically trained *ex-situ* using a software model of the optical system on a digital computer. After training, the voltage drivers in the PhS are configured to realize the trained weights.

Uncertainties in SPNNs

Silicon-photonic integrated circuits are sensitive to nanometer-scale lithographic variations, manufacturing defects, and thermal crosstalk. In this section, we explore the fabrication process variations and run-time uncertainties affecting different photonic components.

Fabrication-process variations in C-SPNNs

Imperfections in the optical lithography process may lead to variations in the resist sensitivity, resist age or thickness, exposure change, and etching. A prominent example of such variation is in the Si waveguide width and thickness. Owing to the high refractive index contrast between the Si core and SiO_2 cladding (Figure 1e), variations in the waveguide width and thickness significantly perturb the effective index. The effective index (n_{eff}) is the ratio between the phase shift per unit length in a waveguide relative to the phase shift per unit length in vacuum. The effective index also depends on the wavelength of the optical signal.

The temperature-dependent phase shift in PhSs is given by

$$\Delta\phi = \left(\frac{2\pi l}{\lambda_0}\right) \cdot \left(\frac{dn}{dT}\right) \cdot \Delta T$$

where l is the length of the PhS and λ_0 is the optical wavelength [3]. Also, $dn/dT \approx 1.8 \cdot 10^{-4}$ is the thermo-optic coefficient of silicon at $\lambda_0 = 1{,}550$ nm and temperature $T = 300$ K, and ΔT is the temperature change. The tuned phase shift $\Delta\phi$ can also change under lithographic variations in l. Additionally, impurities introduced in the waveguide material during fabrication can affect dn/dT.

The microheaters in PhSs are controlled either by applying a tuned voltage or passing a tuned current across the resistor coil. This voltage/current can be supplied from a dc source based on a digital-to-analog converter (DAC). The precision of the temperature shift ΔT and in turn the phase shift are limited by the quantization error in the DAC. For example, in an 8-bit DAC, only 256 different phase shifts in the range $[0, 2\pi]$ can be realized. Low-precision PhSs can degrade the accuracy of the linear multipliers in SPNNs.

The power coupling coefficient in directional-coupler-based BeSs denotes the fraction of input power coupled from one member waveguide to the other. This is given by $K(z) = \sin^2(\delta z)$, where z is the coupler length and δ is the field coupling coefficient. In ideal 50:50 BeS $\left(r = t = 1/\sqrt{2}\right)$, $K(z) = 1/2$. Variations in the waveguide dimensions and the gap between the coupled waveguides arising from proximity effects in the etching process affect δ. Changes in δ, in addition to variations in the coupler length z, can lead to nonidealities in BeSs.

Fabrication-process variations have a significant impact on the individual PhS and BeS; as a result, MZIs are highly sensitive to manufacturing uncertainties. Indeed, MZIs are more sensitive to differential variations among the two constituent waveguides than the common-mode variations to the entire device. This is because the operation of interferometric devices (e.g., MZIs) depends on the phase difference between optical signals in the constituent waveguides. As a result, common-mode variations, which affect optical signal on both the waveguides uniformly, do not have a significant impact on the performance of MZIs. Clearly, understanding the uncertainties in silicon-photonic circuits (including SPNNs) is essential for yield ramp-up.

Run-time uncertainties in C-SPNNs

Run-time uncertainties in C-SPNNs can arise due to mutual thermal crosstalk among the microheaters in thermo-optic PhSs. The tuned phase shift in thermo-optic PhSs is proportional to $l \cdot \Delta T$, where l and ΔT denote the PhS length and the change in temperature, respectively. To minimize the MZI area overhead, larger ΔT is required for tuning PhSs. This necessitates increased heater power consumption and results in higher susceptibility to thermal crosstalk. In fact, even the most efficient PhS requires a voltage $V_\pi = 4.36$ V and power $P_\pi = 24.77$ mW to provide a phase shift of π [8]. The change in phase in the victim PhS due to thermal crosstalk depends on its geometric structure, heater material, and the distance from the aggressor PhS. For a 5-μm aggressor-victim gap filled with the default SiO_2 cladding and $P_\pi = 24.77$ mW, the optical phase shift in the victim PhS is greater than 0.5 rad [3]. Note that due to the latency associated with thermal tuning, the effects of thermal crosstalk may not be localized among proximal microheaters, especially in C-SPNNs with several MZIs. Moreover, due to simultaneous thermal gradients emanating from multiple MZIs, developing a high-fidelity thermal model is complex and requires experimental measurements. Therefore, in our analysis of the impact of thermal crosstalk, we consider crosstalk-induced phase uncertainties in all MZIs.

Prolonged voltage biasing of optical components can lead to the formation of traps at the Si-SiO_2 boundary in optical waveguides. Such traps affect the refractive index of the Si core, thereby leading to higher scattering-induced optical loss. Experimental results on on-chip photonic networks show up to a 30% increase in the energy-delay product due to trap-induced aging. Similar aging-induced run-time uncertainties will also affect C-SPNNs due to long-term thermal biasing.

Hierarchical analysis of the impact of uncertainties in C-SPNNs

While there are different sources of uncertainties in PhSs and BeSs (e.g., lithographic variations, defects, impurities, and thermal crosstalk), their impact can be modeled by considering uncertainties in the phase shifts (for PhSs) and splitting ratios (r and t for BeSs). In this section, we present a case study on the impact of uncertainties in these parameters due to lithographic variations and thermal crosstalk. However, our criticality assessment approach is agnostic to the source of uncertainties and will therefore hold for any other sources of uncertainties affecting the phase shifts and splitting ratios. Figure 1a–d shows the different hierarchical levels in our analysis. Component-level uncertainties in the PhS and BeSs lead to faulty MZI operation at the device level. An array of faulty MZIs lead to deviated matrices (e.g., U, Σ, and V^H); this in turn leads to faulty weight matrix (M) at the layer level. At the system level, a C-SPNN with such faulty weight matrices leads to inferencing errors. We conclude this section with a discussion on few mitigation techniques to improve the tolerance of C-SPNNs against uncertainties.

Component level: PhSs and BeSs

The phase shifts in thermo-optic PhSs can be affected due to lithographic variations in the waveguide, quantization error in the DAC, and thermal crosstalk. Phase uncertainties from these sources can be classified into two main types.

1. *Nominal-dependent (ND) phase uncertainties:* The standard deviation of the phase uncertainties is proportional to the nominal tuned phase shift. In this case, the deviated phase shift is given by $\tilde{\phi} = \phi + \sigma_{\mathrm{nd}}\phi\,\aleph(0,1)$. Here, ϕ and $\aleph(0,1)$ denote the nominal tuned phase shift and the standard normal distribution, respectively. The standard deviation of the uncertainties ($\sigma_{\mathrm{nd}}\phi$) increases with ϕ. ND uncertainties predominantly affect PhSs with high phase shift; typical sources include thermal crosstalk and quantization errors.
2. *Nominal-independent (NI) phase uncertainties:* In this case, the standard deviation of the uncertainties is independent of the tuned phase shift, $\tilde{\phi} = \phi + \sigma_{\mathrm{ni}}\,\aleph(0,1)$. NI uncertainties include geometric process variations in the waveguide and manufacturing defects and impurities.

Prior studies indicate a mean phase uncertainty of up to 0.21 rad ($\approx 0.07\pi$) in fabricated PhSs. To consider a range of uncertainties around this mean, we vary σ_{nd} and σ_{ni} in the range $[0.005\pi,\ 0.15\pi]$. In ideal 50:50 BeSs, $\left(r = t = 1/\sqrt{2}\right)$ (the "Fabrication-process Variations in C-SPNNs" section). However, with uncertainties, a deviation of 1%–2% is typically expected in the r and t parameters. For our analysis, we consider the deviated reflectance $\tilde{r} = r + \sigma_{\mathrm{BeS}}\,\aleph(0,1)$ with the deviated transmittance t $\tilde{t} = \sqrt{(1-\tilde{r}^2)}$. For a fair comparison with the impact of PhS uncertainties, σ_{BeS} is varied in the range $\left[0.005.1/\sqrt{2}, 0.15.1/\sqrt{2}\right]$. Note that uncertainties in the BeS are, in principle, NI as all the devices have the same nominal splitting ratios $r = t = 1/\sqrt{2}$.

Device level: MZIs

Variations in the phase shifts and splitting ratios affect the MZI transfer matrix T_{MZI} (1). To measure the closeness between the deviated transfer matrix $\tilde{T}_{\mathrm{MZI}}$ and T_{MZI}, we use the fidelity metric given by $F(T,\tilde{T}) = \left|\mathrm{Trace}(\tilde{T}\ T)/N\right|^2$. Here, $\tilde{T}^{\dagger}$ and N denote the conjugate transpose and the size of $\tilde{T}$, respectively. Note that $F(T,\tilde{T}) = 1$ if and only if $T = \tilde{T}$ and F decreases with decreasing similarity between T and $\tilde{T}$. Figure 2a shows how F changes due to ND phase uncertainties. In this case, the deviated phase shifts are $\tilde{\theta} = \theta\,(1 + \Delta_{\mathrm{rel}})$ and $\tilde{\phi} = \phi\,(1 + \Delta_{\mathrm{rel}})$, where Δ_{rel} denotes the relative change in the phase shifts. Clearly, an MZI with higher phase shifts is more susceptible to ND phase uncertainties (the z-axis in Figure 2a denotes $1/F$). However, for NI phase uncertainties, F is independent of θ and ϕ. The susceptibility of different

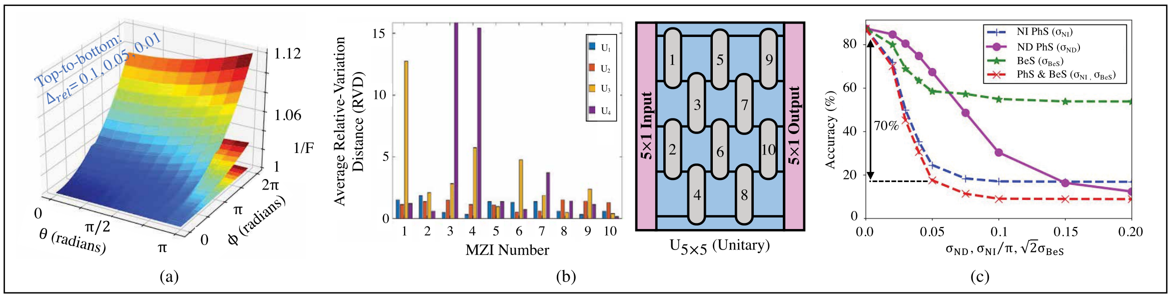


Figure 2. (a) Deviation in T_{MZI} due to ND phase uncertainties. (b) Average RVD (left) for four 5 × 5 unitary matrices with one MZI under variations at a time. Right: MZI array (including the MZI numbers) to represent any 5 × 5 unitary matrix. (c) Impact of ND and NI phase uncertainties and uncertainties in BeS splitting ratio on the C-SPNN inference accuracy.

MZIs to such uncertainties, and also to uncertainties in the splitting ratio, depends solely on their position in the MZI array.

Layer level: MZI array

Unitary multipliers in the linear layers of C-SPNNs can be realized using MZI arrays. Due to faulty MZIs, these unitary multipliers can deviate from their intended form. The deviation can be measured using the relative-variation distance (RVD) given by

$$\mathrm{RVD}\left(\tilde{U},U\right)=\frac{\sum_{m=1}^{N}\sum_{n=1}^{N}\left|\tilde{U}_{m,n}-U_{m,n}\right|}{\sum_{n=1}^{N}\sum_{i=1}^{N}\left|U_{m,n}\right|}.$$

Here, $U(\tilde{U})$ denotes the $N \times N$ intended (deviated) unitary matrix and $|U_{m,n}|$ denotes the absolute value of $U_{m,n}$. Figure 2b shows the mean RVD (over 1,000 iterations) when uncertainties with $\sigma_{\mathrm{nd}} = 0.05$, $\sigma_{\mathrm{ni}} = 0.05\pi$, and $\sigma_{\mathrm{BeS}} = 0.05c$ are inserted in one MZI at a time, in four different randomly generated 5×5 unitary matrices. We observe that the distribution of mean RVD differs across the four unitary matrices. Therefore, the impact of uncertainties in the MZI array on the unitary multipliers depend on both the phase shifts and the position of the affected MZI.

System level: C-SPNN

Incorrect matrix multiplication at the layer level can lead to misclassifications in the C-SPNN. To understand the impact of uncertainties in the phase shifts and splitting ratios on the classification accuracy, we consider an imprecise fully connected C-SPNN with two hidden layers of 16 complex-valued neurons. Each linear layer is followed by a nonlinear Softplus layer. A LogSoftMax layer is used after the output layer to obtain a probability distribution. We use a cross-entropy loss function during training. To reduce the feature vector size, each real-valued MNIST image is converted to a complex feature vector of length 16 using fast Fourier transform [9].

Figure 2c shows the mean inference accuracy (over 1,000 Monte Carlo iterations) under random ND and NI uncertainties in PhSs (characterized by σ_{nd} and σ_{ni}) and uncertainties in BeSs (characterized by σ_{BeS}). We observe that for the different cases, the inference accuracy declines steeply due to these uncertainties. In particular, with uncertainties in both PhSs and BeSs, the accuracy drops by ≈70% even under low levels of uncertainties ($\sigma_{\mathrm{ni}} = 0.05\pi$ and $\sigma_{\mathrm{BeS}} = 0.05/\sqrt{2}$). Also, uncertainties in PhSs have a higher impact on the accuracy compared to similar uncertainties in BeSs.

Understanding the impact of localized uncertainties in the MZI array is necessary for identifying the critical components in an SPNN. The tolerance of an MZI is defined as the maximum allowable change in the splitting ratio of a component BeS that can be recovered using postfabrication thermal tuning in PhSs. Based on this notion of tolerance, it is found that the central MZIs in an array, which require a tuned phase shift very close to 0, have the minimum tolerance to BeS fabrication errors. However, the tolerance of an MZI to uncertainties (or the lack thereof) can also be quantified by the

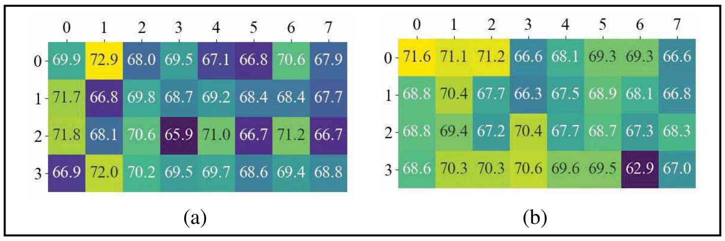


Figure 3. Average accuracy loss (in %) due to zonal perturbations in the unitary weight matrices representing the weights in the first hidden layer. (a) $UL1$. (b) V^{H}_{L1}.

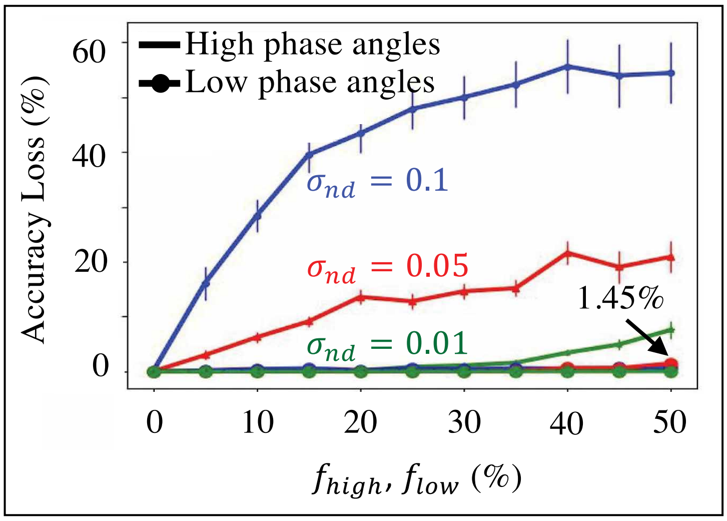


Figure 4. Comparison between the loss in inferencing accuracy in the presence of ND phase uncertainties in the PhS with the top (f_{high}%) and bottom (f_{low}%) phase shifts in each layer.

accuracy loss due to localized uncertainties in the MZI. A higher accuracy loss signifies lower tolerance of an MZI to localized uncertainties. To simulate the impact of localized NI uncertainties, we divide the C-SPNN into zones of four MZIs (in a 2×2 grid). We then select one zone at a time to insert uncertainties with $\sigma_{\text{ni}} = 0.1\pi$ and $\sigma_{\text{BeS}} = 0.1/\sqrt{2}$, while all other zones have background uncertainty with $\sigma_{\text{ni}} = 0.05\pi$ and $\sigma_{\text{BeS}} = 0.05/\sqrt{2}$. Figure 3 shows the mean accuracy loss (over 1,000 Monte Carlo iterations) due to localized uncertainties in the two unitary matrices corresponding to the first hidden layer in our C-SPNN in the form of heatmaps. Each cell in the heatmaps corresponds to a zone with 2×2 MZIs. The value (color) in each cell denotes the accuracy loss due to uncertainties. We observe that even under similar levels of uncertainties, the accuracy loss can vary by up to 10%. Also, note that the low- and high-impact zones are arranged randomly in each heatmap. This reiterates our prior observation that the susceptibility of MZIs to different uncertainties depends on the tuned phase shifts as well as their location in the array.

However, in the presence of ND phase uncertainties, the inferencing accuracy is strongly correlated with the tuned phase shift of the affected MZI(s)—MZIs with higher phase shifts are more susceptible to such uncertainties. To demonstrate this, we rank the tuned phase shifts in each layer of our example C-SPNN in decreasing order, and insert ND uncertainties (quantified by σ_{nd}) to the top f_{high}% and bottom f_{low}% ranked phase shifts. Figure 4 shows the inferencing accuracy loss due to such uncertainties can be catastrophic (up to $\approx 60\%$) when MZIs with higher phase angles are affected. In contrast, MZIs with lower phase angles are practically resilient to ND uncertainties. Therefore, minimizing the tuned phase shifts improves the C-SPNN performance under such uncertainties, in addition to improving their power efficiency (static power consumption in PhSs is proportional to the tuned phase shift). However, in realistic scenarios, C-SPNNs encounter both NI and ND uncertainties and therefore the overall susceptibility of MZIs to uncertainties depends on both their tuned phase shift and location.

Mitigating the impact of uncertainties in C-SPNNs

The extent of the impact of fabrication process and run-time uncertainties on C-SPNNs has only recently been fully understood and as such, there are very few uncertainty mitigation techniques specific to C-SPNNs. Postfabrication trimming approaches can minimize the phase uncertainties between the two arms of an MZI by implanting Ge in the Si waveguide.

Ge implantation converts crystalline Si (lower refractive index) into its amorphous form (higher refractive index) by breaking the chemical bonds. Due to this, the refractive index (and in turn, the phase shift) in each arm can be precisely trimmed by laser annealing [10]. However, postfabrication calibration methods rely heavily on the characterization of individual MZIs; therefore, this method is infeasible for C-SPNNs with high MZI count. To reduce thermal crosstalk, microheaters can be isolated using deep trenches cutting through the SiO_2 cladding. These structures do not involve special fabrication techniques and lead to a 3× reduction in the phase shift under thermal crosstalk [3]. Recent search efforts for mitigation techniques also focus on uncertainty resilient architectures such as the FFTNet which reduces the optical depth and utilizes fewer MZIs, and the diamond topology where the symmetric structure leads to uniform optical losses in each input-to-output path. An uncertainty-aware training method that uses a modified cost function during training and postfabrication hardware calibration is presented in [11]. A novel zero-cost optimization technique that improves the power efficiency and robustness by leveraging the nonuniqueness of singular value decomposition has been proposed in [12].

SPNNs ARE PRONE to nanometer-level fabrication process variations, interdevice thermal crosstalk, optical loss, and manufacturing defects. Each of these sources of uncertainties affects the phase angles and the splitting ratios in different ways. In this article, we have presented a comprehensive analysis of the various fabrication-process variations and run-time uncertainties and explored several methods to mitigate their impact on the performance of an SPNN. We have used a unified hierarchical approach for criticality assessment of these uncertainties and shown that the degradation in performance depends on both the tuned parameter values and the position of the affected components. Our framework can be used for posttraining identification and compensation of critical SPNN components. ■

References

[1] D. A. B. Miller, "Device requirements for optical interconnects to silicon chips," *Proc. IEEE*, vol. 97, no. 7, pp. 1166–1185, Jul. 2009.

[2] A. R. Totović et al., "Femtojoule per MAC neuromorphic photonics: An energy and technology roadmap," *IEEE J. Sel. Topics Quantum Electron.*, vol. 26, no. 5, pp. 1–15, Oct. 2020.

[3] M. Jacques et al., "Optimization of thermo-optic phase-shifter design and mitigation of thermal crosstalk on the SOI platform," *Opt. Exp.*, vol. 27, no. 8, pp. 10456–10471, Apr. 2019.

[4] J. S. Orcutt et al., "Nanophotonic integration in state-of-the-art CMOS foundries," *Opt. Exp.*, vol. 19, no. 3, pp. 2335–2346, Jan. 2011.

[5] M. R. Watts et al., "Adiabatic thermo-optic Mach–Zehnder switch," *Opt. Lett.*, vol. 38, no. 5 pp. 733–735, 2013.

[6] I. A. D. Williamson et al., "Reprogrammable electrooptic nonlinear activation functions for optical neural networks," *IEEE J. Sel. Topics Quantum Electron.*, vol. 26, no. 1, pp. 1–12, Feb. 2020.

[7] S. Khan et al., "Low-loss, high-bandwidth fiber-to-chip coupling using capped adiabatic tapered fibers," *APL Photon.*, vol. 5, no. 5, 2020, Art. no. 056101.

[8] N. C. Harris et al., "Efficient, compact and low loss thermo-optic phase shifter in silicon," *Opt. Exp.*, vol. 22, no. 9, pp. 10487–10493, May 2014.

[9] S. Banerjee, M. Nikdast, and K. Chakrabarty, "Modeling silicon-photonic neural networks under uncertainties," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Feb. 2021, pp. 98–101.

[10] X. Chen et al., "Post-fabrication phase trimming of Mach–Zehnder interferometers by laser annealing of germanium implanted waveguides," *Photon. Res.*, vol. 5, no. 6, pp. 578–582, 2017.

[11] Y. Zhu et al., "Countering variations and thermal effects for accurate optical neural networks," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Nov. 2020, pp. 1–7.

[12] S. Banerjee et al., "Optimizing coherent integrated photonic neural networks under random uncertainties," in *Proc. IEEE/OSA OFC*, Jun. 2021, pp. 1–3.

Sanmitra Banerjee is pursuing a PhD in electrical and computer engineering with Duke University, Durham, NC 27708 USA. His current research interests include fault modeling and design-for-testability solutions for artificial intelligence accelerators based on emerging technologies. Banerjee has a BTech from the Indian Institute of Technology, Kharagpur, India. He is a Student Member of IEEE and ACM SIGDA.

Mahdi Nikdast is an assistant professor with the Department of Electrical and Computer Engineering (ECE), Colorado State University, Fort Collins, CO 80523 USA, where he leads the Electronic-Photonic System Design Laboratory. Nikdast has a PhD in electrical and computer engineering from the Hong Kong University of Science and Technology, Hong Kong. He is a Senior Member of IEEE.

Krishnendu Chakrabarty is the John Cocke Distinguished Professor of Electrical and Computer Engineering at Duke University, Durham, NC 27708 USA. His current research projects include design-for-testability of 3-D ICs, systolic-array and silicon photonic AI accelerators, microfluidic biochips, hardware security, AI for healthcare, and neuromorphic computing systems. Chakrabarty has a PhD from the University of Michigan, Ann Arbor, MI, USA. He is a Fellow of ACM and IEEE. He is also a Golden Core Member of the IEEE Computer Society.

■ Direct questions and comments about this article to Sanmitra Banerjee, Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708 USA; sb535@duke.edu.

Analysis and Mitigation of DRAM Faults in Sparse-DNN Accelerators

Shamik Kundu
The University of Texas at Dallas
Richardson, TX 75080 USA

Kanad Basu
The University of Texas at Dallas
Richardson, TX 75080 USA

Arnab Raha, Suvadeep Banerjee, and Suriyaprakash Natarajan
Intel Corporation, Santa Clara, CA 95054 USA

Editor's notes:
This article analyzes the dynamic random-access memory (DRAM) faults in sparse deep neural network (DNN) accelerators and presents a systematic quality-aware mitigation strategy.
—Fei Su, Intel Corporation

With the increasing size of deep neural networks (DNNs), accelerators fail to efficiently process the large amounts of synaptic weights, thereby incurring intensive computation and memory accesses [1]. Sparse-DNN accelerators aim to alleviate this challenge posed by the overwhelming number of neurons and synapses, while maintaining the accuracy of the original network. Sparse accelerator employs a novel dataflow that enables maintaining the sparse tensors in a compressed encoding format, which minimizes data transfers and reduces storage requirements.

Dynamic random access memory (DRAM) is traditionally utilized as the main memory subsystem in such resource-constrained sparse accelerators. However, due to device-level nonidealities in this semiconductor memory configuration, cells in the DRAM tend to leak out charge, thereby inducing bit-flips in the structure past their retention time. Hence, the DRAM module is periodically refreshed to replenish the cells, thereby incurring considerable power overheads. To address this issue, several researchers proposed lowering the refresh rates [2], [3]. However, this results in bit-flips that are highly dependent on the temperature and VRTs of each cell in the structure. Such bit-flips impacting the crucial model parameters can be detrimental to DNN inference outcomes. Since such accelerators are often deployed in mission-critical environments, mitigating the impact of such faults is imperative to avoid catastrophic circumstances. A point to note here is that, even though the accuracy of the DNN accelerator is of paramount importance in safety-critical applications, the energy consumption of such systems often incurs significant costs in resource-constrained scenarios [4]. Hence, approximation has emerged as a strategy to provide execution performance and save computational resources at the cost of a tolerable precision reduction in high-assurance environments [4], [5]. Depending on the application and the accuracy

Digital Object Identifier 10.1109/MDAT.2022.3183545
Date of publication: 15 June 2022; date of current version: 10 March 2023.

of the DNN accelerator, the designer incorporates approximation in the system performance and defines a conservative threshold, below which the impact of a manifested error is deemed as critical by the deployed application.

In this article, we develop a novel error injection framework that analyzes the impact of errors on application quality. Variations in device-level characteristics of the transistors in the DRAM engender faults throughout the structure, which are induced as bit-flip errors at random locations in the memory subsystem. Moreover, we provide a systematic approach to mitigate faults, thereby improving the reliability in the context of energy-limited sparse-DNN accelerators. To the best of our knowledge, this is the first work that focuses on the reliability of sparse-DNN accelerators.

The key contributions of this article are as follows.

- This article demonstrates a novel error injection framework in the memory subsystem of a sparse-DNN accelerator, which exploits the sparsity in a tensor graph to expedite the execution of network inference.
- Application-level reliability analysis of the accelerator for varying DRAM refresh intervals at multiple operating temperatures, executing multiple DNN architectures on multivariate data sets, provides an estimation of the vulnerability of the architecture.
- Finally, by virtue of the step-by-step systematic mitigation technique, our proposed framework incorporates approximation in the sparse accelerator that maintains the classification accuracy above the critical threshold, while also minimizing the energy consumption by lowering the refresh interval of the memory module to the maximum extent possible.

Related Work

Ongoing research on implementing DNNs at the edge has led to the development of sparse accelerators, which leverages the sparsity of the network architecture to furnish improved performance and energy efficiency over traditional dense accelerators [1]. To further reduce the energy consumption in these resource-constrained environments, existing research has proposed the utilization of suboptimal DRAM refresh rates, thereby approximating the device performance [2], [3]. However, at these suboptimal refresh rates, the impact of DRAM bit-flip errors on safety-critical DNN accelerators has not been well explored.

In a complementary metal–oxide–semiconductor (CMOS)-based DNN accelerator, the susceptibility of the architecture under single-event upsets on the datapath is analyzed on multiple convolutional neural network (CNN) models [6]. To explore the impact of errors on the accuracy, permanent faults are injected into the memory [7] and datapath [8] of a DNN accelerator. However, none of these approaches focus on fault tolerance of sparse-DNN accelerators, arising from suboptimal refresh rates in the DRAM.

Background

DRAM basics

A DRAM is usually organized as multiple collections of a 2-D array of DRAM cells, where each DRAM cell consists of a single access transistor and a single capacitor. The charge stored in this capacitor determines the DRAM cell value. Due to various nonidealities associated with the access transistor, the charge stored in the capacitor leaks away over time (retention time). Hence, a DRAM requires its charge to be periodically replenished using an implicit background refresh operation every 64 ms. This periodic refresh operation usually contributes to a significant amount of DRAM energy overhead as well as a performance bottleneck [3].

To address this challenge, recent works have proposed DRAM to increase the refresh period to values that are orders of magnitude higher than the nominal 64 ms [3]. However, it also results in bit errors at random DRAM sites due to variations associated with the underlying process technology, as shown in Figure 1a. The impact of temperature on bit error rate is drastic, compared to the refresh interval. This can be attributed to the DRAM cell retention time, which decreases exponentially with higher operating temperatures [9]. These DRAM bit errors are of two types: true error and antierror, where a bit value of 1 gets flipped to 0 and vice versa, respectively [9]. The number of their occurrences increases with an increase in refresh period as shown in Figure 1b and significantly exceeds the threshold below which error-correction codes (ECCs) in DRAM can be effective. As depicted in Figure 1c, a large number of bit errors also stem from the phenomenon of variable retention time (VRT) [10]. A DRAM VRT cell exhibits multiple retention times (or states) randomly and

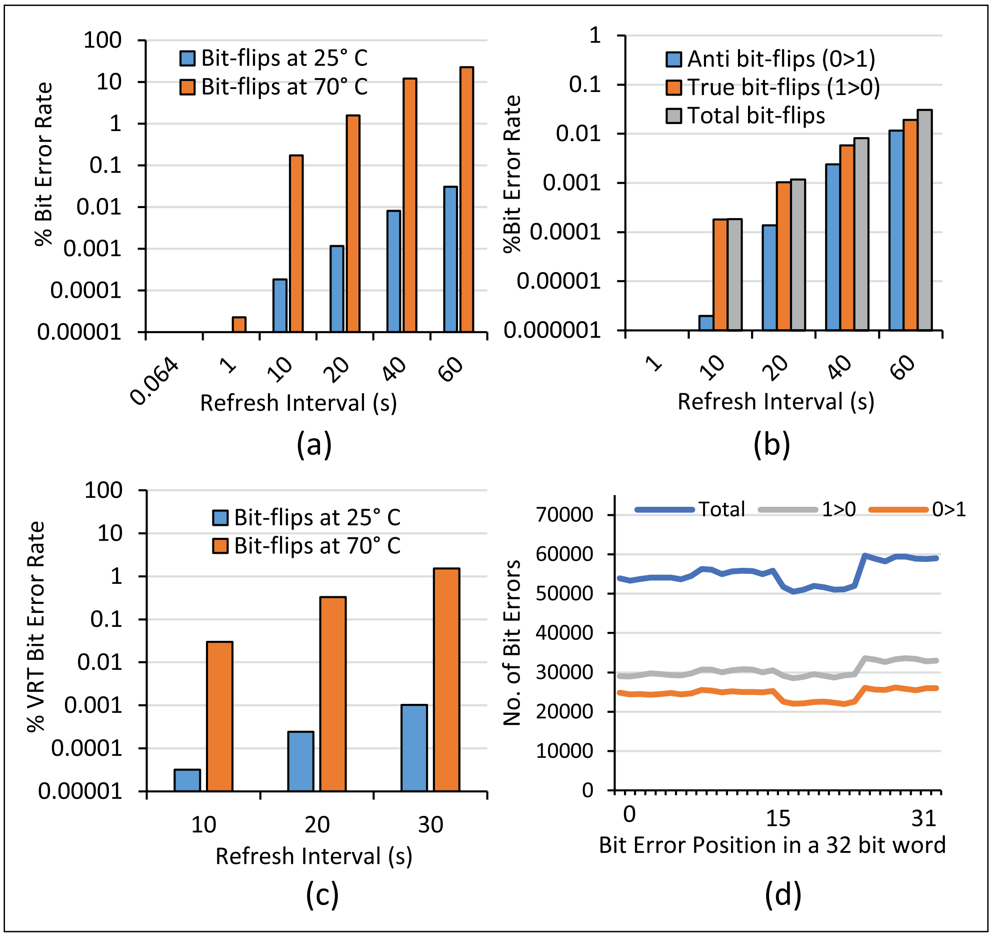


Figure 1. DRAM error characteristics. (a) Variation of refresh bit error rate with temperature and refresh interval. (b) Variation of true and anti-DRAM bit error rate with refresh interval. (c) Increase in VRT cells with refresh interval and temperature. (d) Probability of a bit error occurring at different bit positions. The experimental setup is similar to that described in [3].

unpredictably at different points. The probability of a bit error occurring at different bit positions in a 32-bit word is demonstrated in Figure 1d. The detrimental impact of these errors on DNN accelerators has not been evaluated in existing research, which serves as the primary motivation behind this work. It should be noted that although ECC can address this, it results in additional hardware overhead. Hence, many commercial DRAM models are either devoid of or have limited ECC capabilities [10].

Sparse-DNN accelerators

To accommodate the compute-intensive DNN algorithms, a number of DNN accelerators have been developed. Recently, a new class of sparse DNN accelerators has emerged that accelerates the performance of sparse matrix convolution. Along with the trained network parameters, these accelerators contain compute acceleration and memory compression bitmaps in the DRAM. The compute acceleration bitmap enables the accelerator to leverage the sparsity in a tensor graph and therefore, skip certain computations during the inference, based on the bitmap encoding of the tensors. The memory compression bitmap, on the other hand, allows for reduced data movement and increased on-die storage capacity. Thus, the sparse accelerator furnishes improved performance and energy efficiency by utilizing a dense compressed representation of weights throughout the entire process flow. Since these data and their corresponding bitmaps are stored in the DRAM, their reliability is extremely critical.

Errors in these parameters can lead to a reduction in the classification accuracy of the sparse accelerator. To this end, we propose a novel fault injection framework and subsequently develop a systematic quality-aware mitigation strategy to improve the performance of the sparse accelerator on multiple DNN applications.

Error injection framework

In this article, a DNN is considered to be trained one time in a nonfaulty environment. The trained weights, along with the bitmaps, are stored in the DRAM of the sparse inference accelerator at the edge. In this section, we present a novel error injection that introduces faults in this DRAM architecture at suboptimal refresh rates. A point to note here is that we do not consider the impact of errors in the activations of the DNN, since they are stored in the static random access memory (SRAM) of the accelerator (the next level of memory closer to the processing element), which is much smaller in size and lacks the need of a constant refresh, as opposed to the DRAM. Synaptic weights, on the other hand, are stored for all the layers of the network together in the much larger DRAM and hence are subjected to DRAM faults.

Error in trained model weights

To illustrate the impact of error injection in the pretrained weights, let us consider an output y_i from a particular neuron in a hidden layer of the network, which can be represented as

$$y_i = \sum_j w_{i,j}\, a_j + b_i,\ \ a_i = \phi(y_i) \qquad (1)$$

where $w_{i,j}$ represents a specific weight element from the weight matrix of a particular layer, which is multiplied with activation a_j and subsequently added with bias b_i. The accumulated output over a column of the weight matrix is then passed onto a nonlinear activation ϕ to furnish the output from a distinct neuron in a specific layer of the network. Now, let us consider an error in the DRAM, which when manifested in the stored location of $w_{i,j}$ alters the value by an amount Δw, thereby mapping a weight of $w'_{i,j} = w_{i,j} \pm \Delta w$ instead of $w_{i,j}$ in the inference architecture. Hence, the output y'_i can be represented as

$$\begin{aligned} y'_i &= \sum_j w'_{i,j}\, a_j + b_i = \sum_j (w_{i,j} \pm \Delta w) a_j + b_i \\ &= \sum_j w_{i,j}\, a_j \pm \Delta w a_j + b_i \end{aligned} \qquad (2)$$

As a result of this additional $\Delta w a_j$, y'_i furnishes an erroneous computation, which is propagated to all the neurons in the subsequent layers of the network. The impact of the injected fault in the weight thus amplifies, degrading the classification accuracy.

Error in compute acceleration bitmap

Compute acceleration bitmaps are generated by ANDing the sparsity bitmaps obtained from performing zero-valued compression on both activations and weights. These bitmaps stored in the DRAM enable the sparse accelerator to skip certain computations, thereby expediting the inference execution. For example, let us consider the weights corresponding to a specific layer of the network. The sparse accelerator encodes each nonzero weight within the DRAM as "1"; correspondingly, each zero weight is encoded as "0" in the acceleration bitmap, and subsequently stored in the DRAM. Now, as these weights are multiplied with their corresponding activations, the computation for the weights that are mapped as "0" are skipped, and only those mapped as "1" are executed, thereby accelerating the inference.

As shown in Figure 2a, an antierror in this encoded bitmap flips a "0" to "1." But, since this erroneous "1" in the bitmap corresponds to a zero value of the weight, executing this computation will render a zero output, thus having no impact on the inference. However, this unnecessary computation leads to an increase in energy consumption, along with undesired performance penalties. On the contrary, as a true error in this bitmap flips a "1" to "0," the computation that should have been executed is disregarded in the process, as demonstrated in Figure 2a. This modifies the network architecture, leading to substantial degradation of the accuracy.

Error in memory compression bitmap

Memory compression bitmaps, stored in the DRAM, are leveraged by the sparse accelerator to enable zero-valued compression, thereby reducing memory access. Figure 2b represents an illustrative memory compression bitmap, which is generated by encoding the nonzero values as "1" and zeroes as "0" from the original data array. Prior to inference, the accelerator traverses the bitmap sequentially to access the memory locations that have "1," and skips those which are denoted as "0."

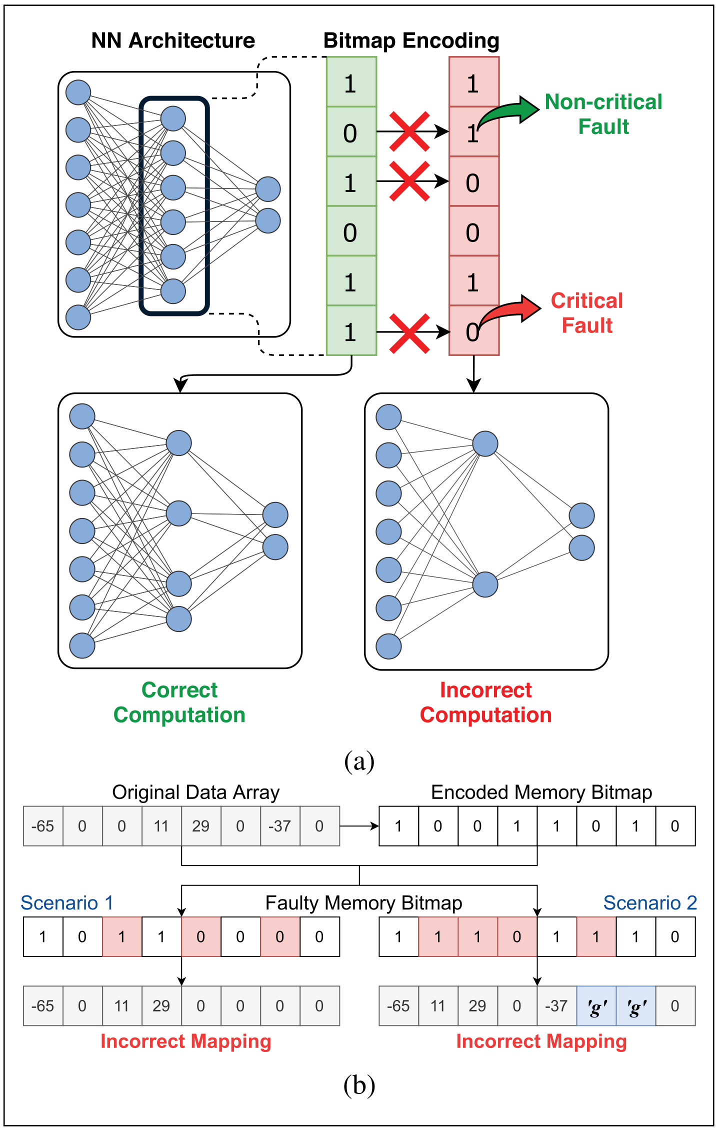


Figure 2. Error injection in (a) compute acceleration bitmap and (b) memory compression bitmap of a sparse accelerator.

If an error (true or anti) is injected in this memory compression bitmap, the mapping of weights will be jeopardized, as demonstrated in Figure 2b. Let us consider scenario one, where the bit-flips at specific locations are marked in red. In this case, the weights are not only mapped to the incorrect locations, but the last nonzero weight "–37" is also not accessed, thereby resulting in an erroneous representation. In the second scenario, the number of 1's in the encoded bitmap has exceeded the number of nonzero values in the original data array. Hence, along with incorrect weight mapping, when all the nonzero values from the original data array have been accessed, a garbage value "g" is dumped in the corresponding location with "1" in the memory compression bitmap. Thus, injecting both true and antierrors leads to an erroneous representation of the weight array, thereby modifying the network configuration, resulting in degradation in performance in the sparse accelerator. For cases where the compute acceleration bitmap is not stored in DRAM but computed internally during inference, a fault in the memory compression bitmap translates to a fault in compute acceleration bitmap as well.

Experimental results

Experimental setup

We considered two popular neural network architectures, multilayer perceptrons (MLPS) and CNNs, LeNet and VGG-16. MLPs are executed on data sets—Modified National Institute of Standards and Technology (MNIST) and fashion-MNIST (F-MNIST), whereas LeNet is executed on F-MNIST and VGG-16 on CIFAR-10. The MLP has four fully connected layers of (256 – 256 – 256 – 10) configuration. To perform a behavioral simulation of a sparse accelerator, we implemented the MLP and CNNs in Keras and PyTorch frameworks, respectively, using Python3. The trained weights from each network are extracted and quantized to 8 bits to be stored in the DRAM, similar to [8]. In a nonfaulty environment, MLP furnishes a baseline accuracy of 97.28% and 88.17% for MNIST and F-MNIST, respectively. LeNet, on the other hand, furnishes a baseline classification accuracy of 92.16% for F-MNIST, and VGG-16 exhibits a corresponding baseline of 91.43% on CIFAR-10. A point to note here is that even though a model trained with dropout layers is traditionally presumed to improve the resiliency of the DNN against errors, the threshold of such resilience can also be easily inflicted with bit-level faults and hence not considered a baseline. The novel error injection framework performs ten Monte Carlo simulations, in accordance with existing research [11], to induce both true and antierrors at random locations in the weights as well as the encoded bitmaps throughout the DRAM structure. We tracked the errors at a page granularity, where each page acquires a size of 4 KB in a 1 GB DRAM.

Results

Memory compression bitmap under DRAM faults

Figure 3a represents the degradation in accuracy for incremental faults in the memory compression bitmaps of the sparse accelerator. With a single fault, MLP on MNIST furnishes a significant accuracy drop of 80.28%, which increases to a maximum degradation of 88.79% for five faults per layer of the network. An identical trend is observed for the remaining data sets and networks. Therefore, it can be inferred without loss of generality that errors in memory compression bitmaps are detrimental to the reliability of the sparse accelerator.

Compute acceleration bitmap under DRAM faults

The impact on accuracy for both MLP and CNNs is observed by injection of true errors in the encoded compute acceleration bitmap, as represented in Figure 3b and c, respectively. With incremental faults in the bitmap, the degradation in classification accuracy increases to a point, from where it saturates. The saturation point is obtained upon inflicting all the encoded weights with true errors in the last layer of the network. MNIST and F-MNIST furnish maximum accuracy drops of 87.54% and 78.17%, respectively, for a fault rate of 1%. LeNet and VGG-16 on F-MNIST and CIFAR-10 manifest a maximum of 83.11% and 82.36% degradation on the accuracy, respectively, at a 2% fault rate. Hence, errors in the compute acceleration bitmaps also cause significant degradation in network performance.

Model weights under DRAM faults

The position of the induced bit-flip is varied across the three most significant bit positions, starting from the sign bit. The corresponding reduction in accuracy for MLPs on MNIST and F-MNIST have been outlined in Figure 4a and b, while Figure 4d and c demonstrates the performance degradation of CNNs on the F-MNIST and CIFAR-10 data sets, respectively. As the significance of the bit position diminishes, the sensitivity of the induced fault reduces. However, the number of faults required to obtain a significant accuracy drop for faults in model weights is substantially higher than the encoded bitmaps. Hence, the model weights are inferred to be the most resilient, which motivates us to explore the impact of VRT and operating temperature on the weights of the sparse accelerator.

Impact of VRT of DRAM on network performance

In this experiment, the susceptibility of the sparse accelerator under faults is analyzed for VRTs in each cell of the DRAM module at suboptimal refresh rates for a particular operating temperature of 25 °C. The number of bit flips corresponding to each refresh interval is obtained from the DRAM characteristics as demonstrated in Figure 1. Figure 4e outlines the reduction in accuracy for errors induced in the trained weights. The reduction in accuracy increases with incremental refresh intervals until 40 seconds, after which every single weight in the DRAM render to be faulty to furnish a consistent degradation in accuracy. Additionally, medium refresh intervals impact all classes in the data set owing to random fault manifestation in each of the ten runs. As a result, a slightly increased accuracy drop is exhibited, which furnishes a biased impact of faults on a particular class of data. Hence, as seen from Figure 4e, lower refresh intervals (less than 5 s, but higher than the nominal refresh interval of 64 ms) are imperative to evade degradation in accuracy, thereby bolstering the reliability of the sparse accelerator.

Impact of DRAM temperature on network performance

To analyze the impact of variation in temperature on bit errors, as explained in the previous section, the previous experiment is repeated, but at an operating temperature of 70 °C. While the former simulates a room-temperature environment, the latter indicates a higher temperature that a typical commercial application can attain during operation. The vulnerability of the sparse accelerator for four different networks is observed at 70 °C and plotted alongside 25 °C in Figure 4e. As observed from the figure, at higher refresh intervals, the reduction in classification accuracy saturates for both the temperatures, similar to the previous section.

Mitigation strategy

To mitigate these impacts of faults, we propose a systematic step-by-step quality-aware mitigation technique that incorporates approximation in the

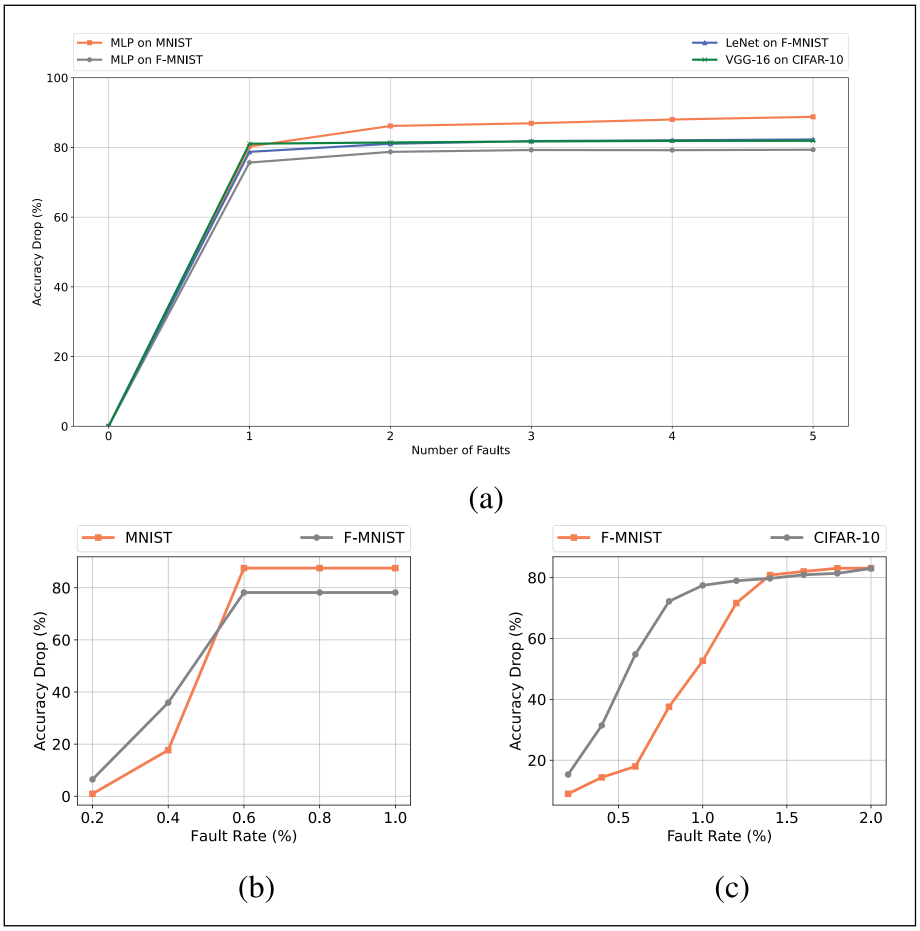


Figure 3. Reduction in classification accuracy for faults in (a) memory compression bitmap and compute acceleration bitmap of (b) MLP on MNIST and F-MNIST; (c) LeNet on F-MNIST and VGG-16 on CIFAR-10.

sparse accelerator. This enables the accelerator to maintain the classification accuracy above the critical threshold, while also minimizing the energy consumption by lowering the refresh interval of the memory module to the maximum extent possible. Prior research [3] has shown that the page error rate in a DRAM due to high refresh intervals follows almost a similar trend as the bit error rate, which is demonstrated in Figure 1a. In such a DRAM, even at higher refresh intervals, there exist certain pages which manifest zero error. We refer to a collection of these zero error pages as an error-free bin.

Utilizing this DRAM configuration, we propose a strategy that is capable of incrementally enhancing the fault tolerance of the sparse accelerator. Depending on the desired performance from the target DNN application, we develop a three-stage fault mitigation technique, as discussed below.

- *Stage I*: The memory compression bitmaps are the first to be allotted in the limited number of error-free bins in the DRAM. Since the memory compression bitmaps are rendered to be the most vulnerable as shown in the previous section, these are prioritized over the compute acceleration bitmaps and the model weights to be stored in the zero-error pages. When this stage is implemented, the accelerator is able to recover up to 76.24% of the classification accuracy of the sparse DNN, averaged over the four different case studies. The worst-case scenario can appear from faults in the compute acceleration bitmaps, which is demonstrated in the previous section.

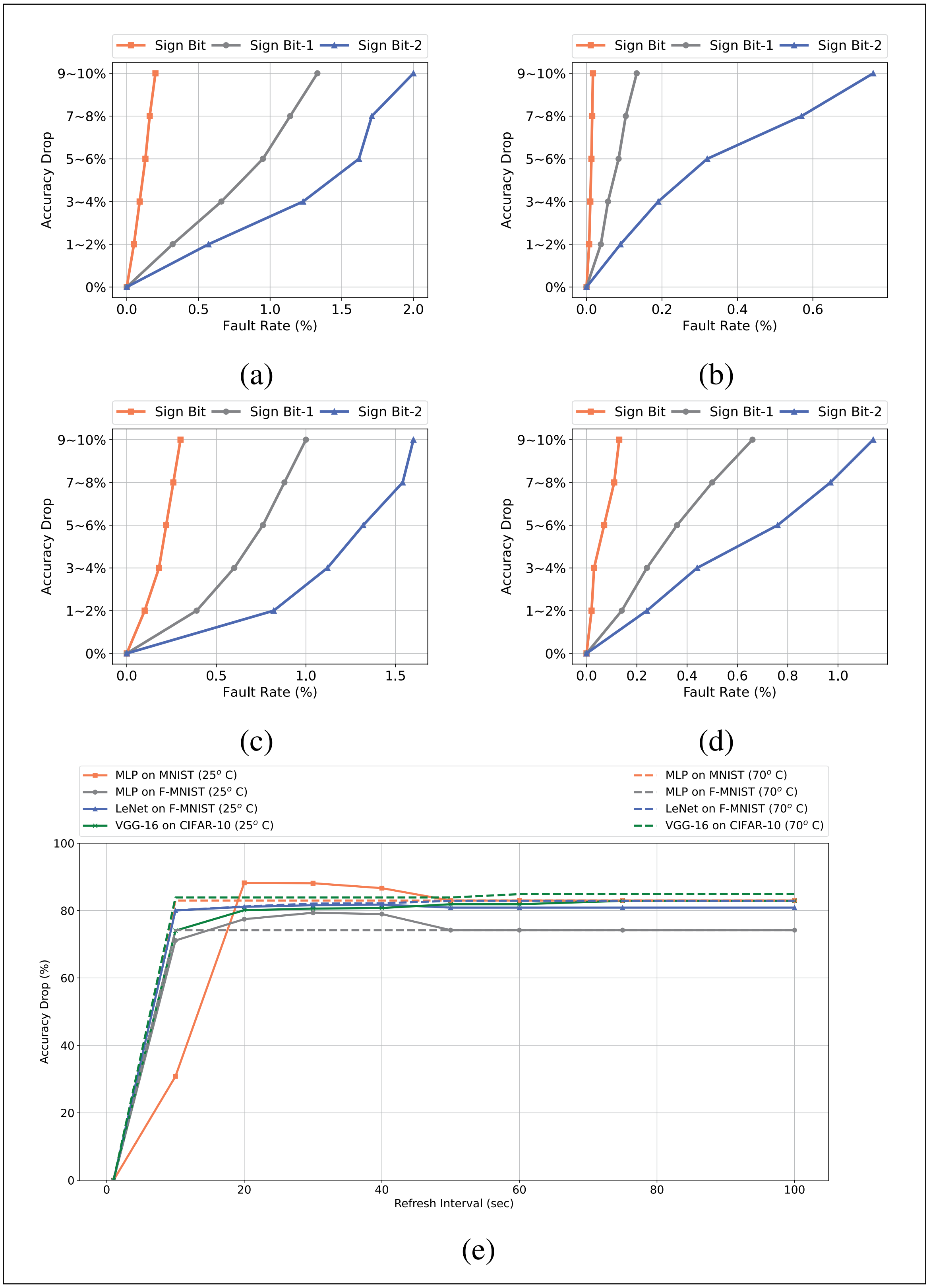


Figure 4.Variation of accuracy drop for faults in model weights of (a) MLP on MNIST, (b) MLP on F-MNIST, (c) LeNet on F-MNIST, and (d) VGG-16 on CIFAR-10. (e) Impact of VRT and DRAM temperature on the classification accuracy of the sparse accelerator.

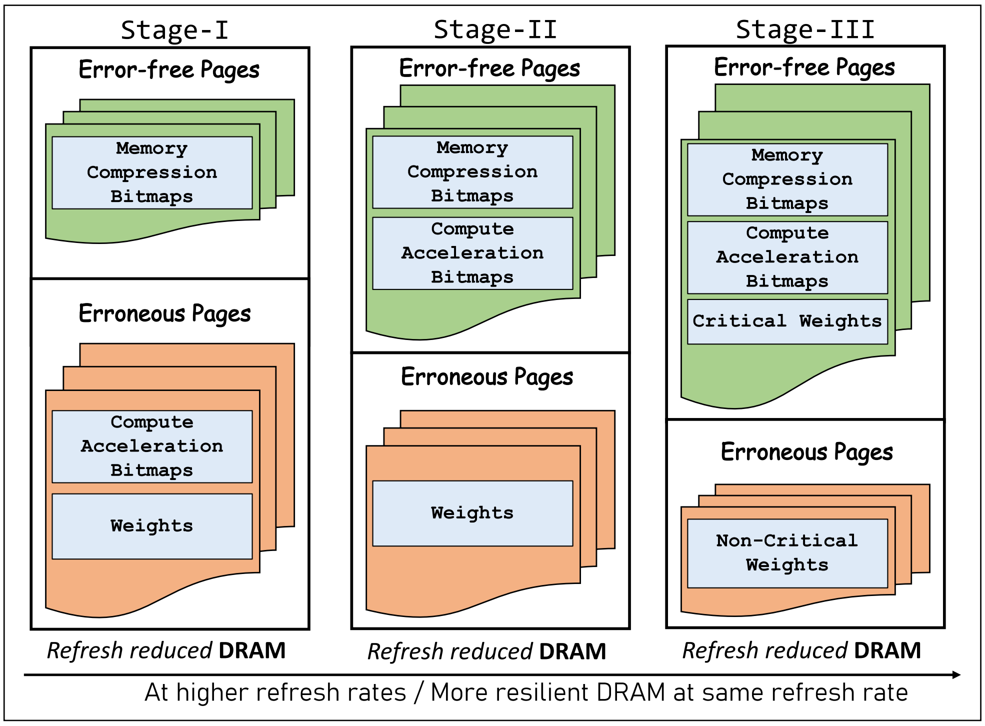

Stage-I
Error-free Pages
Memory Compression Bitmaps
Erroneous Pages
Compute Acceleration Bitmaps
Weights
Refresh reduced DRAM
Stage-II
Error-free Pages
Memory Compression Bitmaps
Compute Acceleration Bitmaps
Erroneous Pages
Weights
Refresh reduced DRAM
Stage-III
Error-free Pages
Memory Compression Bitmaps
Compute Acceleration Bitmaps
Critical Weights
Erroneous Pages
Non-Critical Weights
Refresh reduced DRAM
At higher refresh rates / More resilient DRAM at same refresh rate

Figure 5. Overview of the proposed mitigation strategy.

- *Stage II*: If the DRAM is left with more error-free bins, the encoded compute acceleration bitmaps are then stored in these pages, along with the memory compression bitmaps. This furnishes up to 84.48% recovery in classification accuracy of the sparse DNN accelerator. In this case, faults can manifest in the trained model weights, the impact of which is demonstrated in the previous section.
- *Stage III*: This stage furnishes the storage of model weights along with the encoded memory compression and computes acceleration bitmaps in the available error-free pages in the DRAM. Thus, the sparse accelerator obtains a maximum fault tolerance with 100% recovery in classification accuracy, when exposed to errors in the memory subsystem.

This quality-aware mitigation approach is outlined in Figure 5. This entire operation is performed by the designer, prior to the deployment of the memory module at the edge. Based on the expected accuracy from the target DNN inference, the errors in the vulnerable components can be mitigated with the proposed technique. With the DRAM specifications as mentioned in the previous section, this approach will furnish a minimal memory overhead in the sparse DNN accelerator.

Discussion

In this article, a novel error injection framework is proposed in the context of sparse accelerators. Faults manifested in the DRAM of a sparse DNN accelerator not only incorporate errors in the trained model parameters, but also impact the memory encoding bitmaps and the compute acceleration bitmaps of the sparse network. Faults in the bitmaps inflict the control flow of the accelerator, thereby modifying the topology of the underlying DNN, as demonstrated in the previous section. In comparison, faults in the model weights only introduce errors in computation, without impacting the network architecture. As a result, the reduction in classification accuracy for faults in the sparsity bitmaps is significantly higher compared to faults in the network parameters, as outlined in the previous section.

Following this, we developed a quality-aware fault mitigation technique to incrementally improve the performance of the accelerator, at the cost of minimal memory overhead. In 1-GB DRAM with a page size of 4 KB, the total number of pages is 1 GB/AKB = 262,144. The number of bits required to index each of these pages to identify the error-free ones can be calculated as /oy%262111 = 18. Therefore, an excess of 18 × 262,144 = 4,718,592 bits is required for the mitigation strategy, equivalent to 0.05% memory overhead on the 1-GB DRAM. Thus, in lieu of minimal overhead, this scheme will enable the DRAM to operate at lower refresh intervals, while improving the reliability of the sparse accelerator.

To the best of our knowledge, this is the first work that performs an application-level reliability analysis on a sparse-DNN accelerator subjected to DRAM faults. The analysis derived in this article can be extrapolated to other network architectures as well to explore model-specific fault tolerance. This will aid in storing the critical weights in the error-free bins of the DRAM, where the number of zero-error pages is limited. This will reduce the memory overhead, while improving the reliability of such accelerators used in mission-critical systems. ■

Acknowledgments

This work was supported by the Semiconductor Research Corporation (GRC Task: 2930.001).

References

[1] S. Zhang et al., "Cambricon-X: An accelerator for sparse neural networks," in *Proc. IEEE/ACM MICRO*, Oct. 2016, pp. 1–12.

[2] S. Liu et al., "Flikker: Saving DRAM refresh-power through critical data partitioning," in *Proc. ASPLOS*, 2011, pp. 213–224.

[3] A. Raha et al., "Quality configurable approximate DRAM," *IEEE Trans. Comput.*, vol. 66, no. 7, pp. 1172–1187, Jul. 2017.

[4] G. S. Rodrigues, F. L. Kastensmidt, and A. Bosio, "Approximate computing for safety-critical applications," in *Proc. IEEE 22nd Latin Amer. Test Symp. (LATS)*, Oct. 2021, pp. 1–3.

[5] G. S. Rodrigues et al., "Performances VS reliability: How to exploit approximate computing for safety-critical applications," in *Proc. IEEE 24th Int. Symp. On-Line Test. Robust Syst. Design (IOLTS)*, Jul. 2018, pp. 291–294.

[6] G. Li et al., "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," in *Proc. SC*, Nov. 2017, pp. 1–12.

[7] B. Reagen et al., "Ares: A framework for quantifying the resilience of deep neural networks," in *Proc. ACM DAC*, 2018, pp. 1–6.

[8] S. Kundu et al., "Toward functional safety of systolic array-based deep learning hardware accelerators," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 3, pp. 485–498, Mar. 2021.

[9] J. Liu et al., "An experimental study of data retention behavior in modern DRAM devices: Implications for retention time profiling mechanisms," *ACM SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 60–71, 2013.

[10] M. K. Qureshi et al., "AVATAR: A variable-retention-time (VRT) aware refresh for DRAM systems," in *Proc. IEEE DSN*, Jun. 2015, pp. 427–437.

[11] J. J. Zhang et al., "Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator," in *Proc. IEEE VTS*, Apr. 2018, pp. 1–6.

Shamik Kundu is pursuing a PhD with the Department of Electrical and Computer Engineering, The University of Texas at Dallas, Richardson, TX 75080 USA. He is a Student Member of IEEE.

Arnab Raha is a staff research scientist at the Advanced Architecture Research Group, Intel Edge AI, Santa Clara, CA 95054 USA. Raha has a PhD in electrical and computer engineering from Purdue University, West Lafayette, IN, USA. He is a member of IEEE.

Suvadeep Banerjee is a research scientist with Intel Labs, Santa Clara, CA 95054 USA. Banerjee has a PhD in electrical and computer engineering from the Georgia Institute of Technology, Atlanta, GA, USA. He is a member of IEEE.

Suriyaprakash Natarajan is a senior researcher with Manufacturing and Product Engineering (MPE), Intel Corporation, Santa Clara, CA 95054 USA. Natarajan has a PhD in computer engineering from the University of Southern California, Los Angeles, CA, USA. He is a Senior Member of IEEE.

Kanad Basu is an assistant professor at the Electrical and Computer Engineering Department, The University of Texas at Dallas, Richardson, TX 75080 USA. Basu has a PhD from the Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL, USA. He is a Senior Member of IEEE.

■ Direct questions and comments about this article to Shamik Kundu, The University of Texas at Dallas, Richardson, TX 75080 USA; shamik.kundu@utdallas.edu.

On the Mitigation of Read Disturbances in Neuromorphic Inference Hardware

Ankita Paul, Shihao Song, Twisha Titirsha, and Anup Das
Department of Electrical and Computer Engineering
Drexel University
Philadelphia, PA 19147 USA

Editor's notes:
This article presents a system software framework to mitigate read disturb failures of nonvolatile memory (NVM) cells used to implement synapses in neuromorphic fabric.

—Fei Su, Intel Corporation

Neuromorphic systems are integrated circuits designed to mimic the neural architecture in primates. Here, neural circuity is tightly coupled with synaptic storage, which eliminates the performance and energy bottlenecks of shared-memory systems for machine learning inference [1]. Non-volatile memory (NVM) cells such as oxide-based resistive switching random access memory (OxRRAM) can implement multilevel analog operations, which make them ideal candidates for storing model parameters, that is, the synaptic weights of a machine learning model [2].

For use as an inference hardware, trained model parameters are programmed as resistance states on OxRRAM cells of the hardware. Once programmed, the hardware is expected to perform inference continuously, without having to reprogram the model parameters. Unfortunately, OxRRAM cells suffer from the read disturb issue, where a cell's resistance state may drift from its programmed value upon repeated access during inference [3]. We show that resistance drifts can lead to a lower inference accuracy.

One system-level technique to mitigate read disturbances in a neuromorphic hardware is to periodically reprogram the trained parameters to the OxRRAM cells of the hardware. Reprogramming of model parameters involves transferring the synaptic weights from the main memory (primary storage) to the neuromorphic hardware via bandwidth-limited memory channels (see Figure 11). Additionally, NVM cells require the long-latency program-and-verify (P&V) scheme to configure their resistance states [4]. These factors increase the time it takes to reprogram model parameters on OxRRAM cells. When a model is being reprogrammed, the hardware is unavailable to perform inference operations. Therefore, the performance overhead associated with periodic reprogramming is

$$\text{reprogram overhead} = \frac{t\text{RPT}}{t\text{RPI}} \tag{1}$$

where tRPT defines the reprogramming time of the model and tRPI defines the interval at which the model is being reprogrammed to the hardware.

Digital Object Identifier 10.1109/MDAT.2022.3148967
Date of publication: 2 February 2022; date of current version: 10 March 2023.

We show that periodic reprogramming leads to a high system overhead even for smaller models like LeNet and AlexNet and is expected to become a critical performance bottleneck for emerging large models such as VGGNet, ResNet, and DenseNet. Our *objective* is to minimize this overhead by increasing the reprogram interval *t*RPI. To this end, we make the following three key observations.

Observation 1: Different synaptic connections of a machine learning model have different tolerance to resistance drift and they impact model accuracy differently.

Observation 2: OxRRAM cells in a neuromorphic hardware exhibit variation in read disturbance due to a difference in the exposed read voltage used during inference.

Observation 3: Activation of a synaptic connection in a model is workload-dependent and it leads to a difference in the amount of resistance drift within the model.

Based on these three observations, we propose a system software framework that incorporates this application and voltage-dependent characteristics of read disturbance of OxRRAM cells in implementing a machine learning model on the hardware. The *key idea* is to implement the synaptic weight of connections that have higher activations and lead to higher accuracy drop on NVM cells that are exposed to lower voltages during inference. In this way, we are able to sustain larger resistance drifts of synaptic weights before reprogramming of model parameters on OxRRAM cells becomes necessary.

A preliminary version of this system software framework is proposed in our prior work [5]. Here, we extend this framework in four key directions: 1) introducing overhead due to reprogramming of model parameters as a key performance metric; 2) extending the system software framework to periodically reprogram model parameters to a neuromorphic hardware to maintain integrity of machine learning tasks; 3) a convex optimization formulation of cluster mapping to crossbar to reduce the system overhead; and 4) exploiting machine learning model characteristics to identify non-critical model parameters and eliminating them from the critical path of deciding the reprogramming interval. In this way, our convex optimizer is able to increase the reprogramming interval compared to [5], thereby significantly reducing the system overhead.

We integrate the proposed system software framework inside NeuroXplorer [6], a cycle-accurate simulator of neuromorphic hardware and evaluate it using five commonly used machine learning inference applications. Results demonstrate an average 35% reduction of system overhead.

Resistance drift tolerance of machine learning workloads

Synaptic connections of a machine learning workload have varying tolerances to resistance drift. This impacts accuracy differently. To illustrate this, we consider 2-bit quantized versions of five commonly used convolutional neural networks (CNNs)—LeNet (1989), AlexNet (2012), VGGNet (2015), ResNet (2015), and DenseNet (2017). There are three weight levels used in these models, corresponding to ternary values of –1, 0, and +1 [7]. Figure 1 illustrates the fraction of total synapses in the fully connected layer that leads to 1% or higher accuracy drop. We report results for the following four configurations: 1) resistance reduction by two levels ("–2"); 2) resistance reduction by one level ("1"); 3) resistance increase by 1 level ("+1"); and 4) resistance increase by two levels ("+2").[1] We make the following three key observations.

[1] We note that if a synaptic weight is +1, then the synapse is tolerant to resistance drifts in the positive direction. Similarly, if a synaptic weight is –1, then the synapse is tolerant to drifts in the negative direction. Such cases are included in the results of Figure 1.

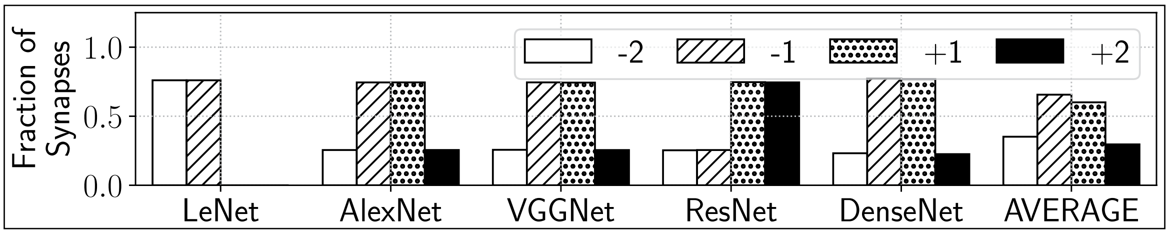


Figure 1. Fraction of total synapses in the fully connected layer that leads to accuracy drop due to resistance drift.

First, synapses in a machine learning model have varying tolerance to resistance drift. On average, only 35% of synapses show accuracy drop when their resistance drifts by two levels in the negative direction, only 66% when resistance drifts by one level in the negative direction, only 60% when resistance drifts by one level in the positive direction, and only 30% when resistance drifts by two levels in the positive direction. The reason for such variations is twofold. First, most machine learning models are over-parameterized. Therefore, resistance levels of noncritical synapses do not impact the accuracy. Second, due to the approximate training using the backpropagation algorithm, a drift in the resistance level of some synapses may not impact accuracy significantly. To this end, we note that the synaptic weight value of a noncritical synapse may not necessarily be close to zero. It simply means that any change of its weight value may not impact accuracy. For this reason, any neuron and synapse pruning strategy such as [7] will not eliminate noncritical synapses that are nonzero.

Second, for LeNet, only a small fraction (less than 1%) of synapses lead to accuracy drop when resistance drifts by +1 and +2. This is because most of synaptic weights of LeNet are positive. So, any transition in the positive direction results in no significant accuracy impact.

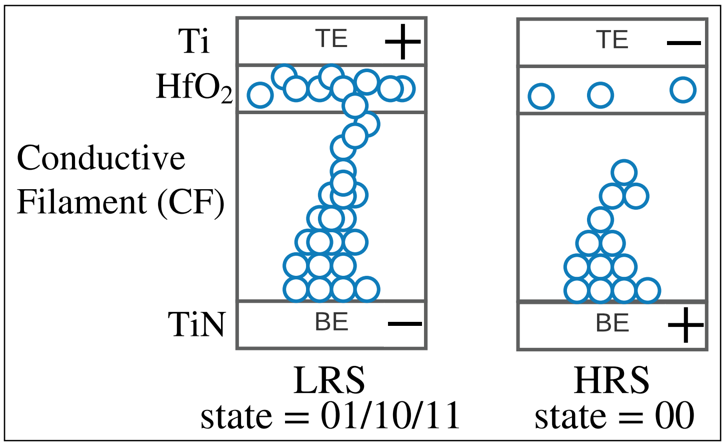


Figure 2. Operation of an OxRRAM cell with the HfO_2 layer sandwiched between the metals Ti (top electrode) and TiN (bottom electrode). The left subfigure shows the formation of LRS states with the formation of CF. The right subfigure shows the depletion of CF on application of a negative voltage on the TE.

Third, tolerance to resistance drift depends on the specific CNN model and therefore, model-specific solutions are needed. Our proposed approach is the following. First, we identify the critical synapses, that is, those that have high impact on accuracy by analyzing a CNN model. Next, we exploit device characteristics and mapping alternatives to minimize the negative impact of resistance drift.

To motivate our solution, we discuss resistance drift in NVMs, focusing on OxRRAM devices.

Resistance drifts in NVM devices

OxRRAM technology

The OxRRAM technology presents an attractive option for implementing the synaptic cells of a crossbar due to its demonstrated potential for low-power multilevel operation and high integration density [2]. An OxRRAM cell is composed of an insulating film sandwiched between conducting electrodes forming a metal–insulator–metal (MIM) structure (see Figure 2). Recently, filament-based metal-oxide OxRRAM implemented with transition-metal-oxides such as HfO_2, ZrO_2, and TiO_2 has received considerable attention due to their low power and CMOS-compatible scaling.

Synaptic weights are represented as conductance of the insulating layer within each OxRRAM cell. To program an OxRRAM cell, elevated voltages are applied at the top and bottom electrodes, which rearranges the atomic structure of the insulating layer. Figure 2 shows the high-resistance state (HRS) and the low-resistance state (LRS) of an OxRRAM cell. An OxRRAM cell can also be programmed into intermediate LRSs, allowing its multilevel operations.

Read disturbance issues of OxRRAM cells

In OxRRAM technology, the transition from HRS to one of the LRS states is governed by a sudden decrease of the vertical filament gap on application of a stress voltage during spike propagation [3]. This is illustrated in the left subfigure of Figure 3, where the vertical filament gap is shown to reduce by an amount *h*. This may result in a conducting filament (CF) between the two metal layers causing the resistive state to change from HRS to LRS. The rate of change of the filament gap of the OxRRAM cell is

$$\frac{dg}{dt} = -\vartheta_0 \cdot e^{-\frac{E_a}{kT}} \sinh\left(\frac{\gamma \cdot a_0}{L} \cdot \frac{qV}{kT}\right), \text{ where } \gamma = \gamma_0 - \beta \cdot \frac{g}{g_0}^3 \cdot \tag{2}$$

In the above equation, t defines the state transition time, g_0 is the initial filament gap of the OxRRAM cell, V is the voltage applied to the cell, γ is the local field enhancement factor, which is related to the gap g, a_0 is the atomic hoping distance, E_a is the activation energy, k is the Boltzmann's constant, T is the temperature (in Kelvin), L is the length of the vertical filament, q is the filament charge, ϑ_0 is a constant related to vertical filament growth, and γ_0 and β are fitting constants.

The transition from one of the LRS states is governed by the lateral filament growth [3]. This is illustrated in the right subfigure of Figure 3. The time for state transition in the OxRRAM cell is given by

$$t(\text{LRS}) = 10^{-14.7 \cdot V + 6.7}\,\text{sec.} \tag{3}$$

If the state transition time of an OxRRAM cell is 1,000 ms, then a single quasi-static read operating using one 1,000 ms read pulse or equivalently, 1,000 read accesses using 1-ms spike pulses can lead to an abrupt change in the cell's state.[2]

From (2) and (3), we see that the state transition time of an OxRRAM cell depends heavily on the voltage of operation of the cell. To this end, we investigate the internal architecture of a processing core in a neuromorphic hardware. In many recent designs, analog crossbars are used as cores. Figure 4 (left) shows an $N \times N$ crossbar, where the OxRRAM cells are organized in a 2-D grid with horizontal wordlines and vertical bitlines. Presynaptic neurons are mapped along wordlines and postsynaptic neurons along bitlines, as shown in the figure. The synaptic weight between a presynaptic neuron (n_i, placed on the ith wordline) and a postsynaptic neuron (n_j, placed on the jth bitline) is programmed as conductance of the OxRRAM cell (i, j) located at the intersection of the ith wordline and jth bitline.

For forward propagation of neuron excitation, a spike voltage is created by n_i, which generates a current that propagates to the neuron n_j via the conductance of the (i, j)th OxRRAM cell. Figure 4 (right) shows the parasitic components on such current paths. Formally, the number of parasitic components on the current path via the (i, j)th OxRRAM cell is $(i + j + 1)$.

Parasitic components on bitlines and wordlines of a crossbar create variation in currents propagating via different OxRRAM cells of the crossbar; the higher the number of parasitic components, the smaller is the current, and vice versa. Therefore, the current through (0, 0)th OxRRAM cell is higher than (N – 1, N – 1)th OxRRAM cell in an $N \times N$ crossbar.

Figure 5 shows the difference between currents on the shortest and longest path for 32 × 32, 64 × 64, 128 × 128, and 256 × 256 crossbars at a 65-nm process node. The input spike voltage of the presynaptic neurons is set to generate 50 μA on the longest path. This current value corresponds to the current needed to read the resistance state of the OxRRAM cell on this path. We observe that the current on the longest path is lower than the shortest path by 13.3% for 32 × 32, 25.1% for 64 × 64, 39.2% for 128 × 128, and 55.8% for 256 × 256 crossbar.

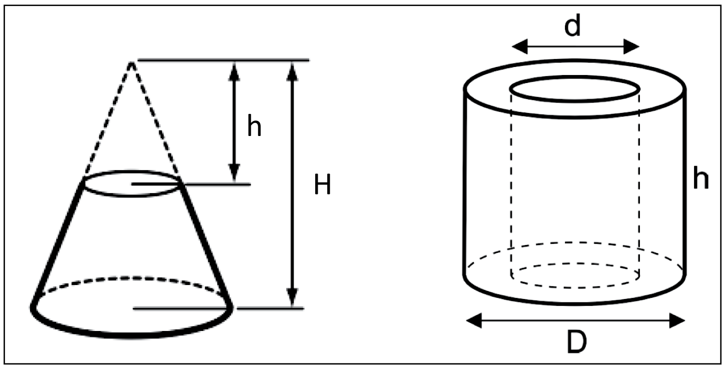


Figure 3. Read disturbances due to structural alteration in an OxRRAM cell. The left subfigure shows a reduction of the conductive filament gap (i.e., read disturbance of HRS state) on the application of a stress voltage. The right subfigure shows the lateral growth of the conductive filament (i.e., read disturbance of LRS state) due to application of a stress voltage.

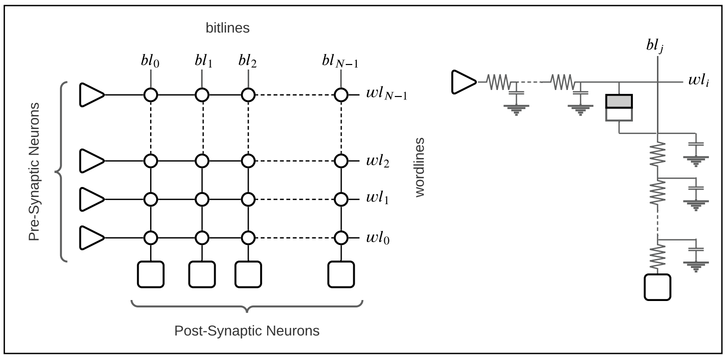


Figure 4. *N* × *N* crossbar showing the parasitic components within.

[2] Apart from resistance drift, there are also other forms of reliability issues reported for OxRRAM in the context of neuromorphic hardware [8]–[11].

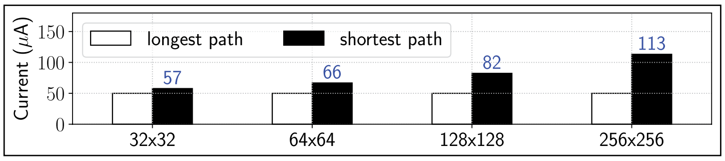


Figure 5. Difference between current on the shortest and longest paths in a crossbar for different crossbar sizes.

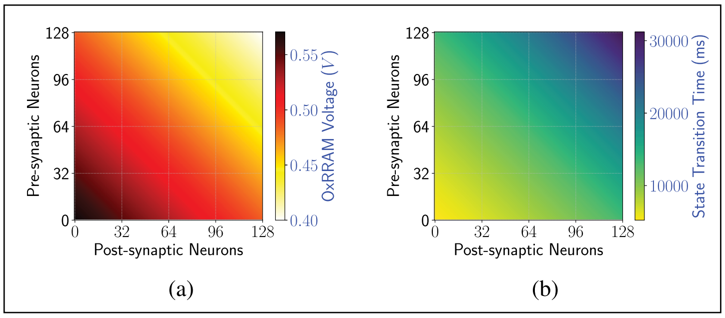


Figure 6. Variation of voltage and state transition time in a 128 × 128 crossbar. (a) Voltage variation in a 128 × 128 crossbar at a 65-nm node. (b) State transition times in a 128 × 128 crossbar with all cells programmed to the HRS state.

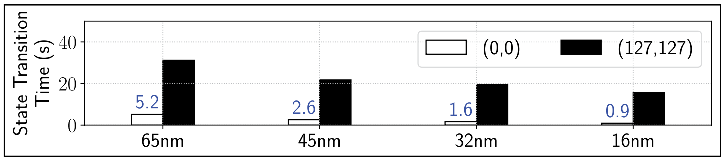


Figure 7. Variation in state transition time of OxRRAM cells in a crossbar as a function of current.

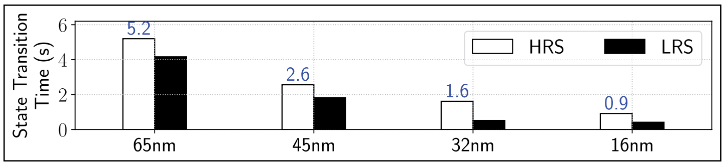


Figure 8. Variation in state transition time of OxRRAM cells in a crossbar as a function of the resistance state.

Current variation in a crossbar leads to a difference in the voltage applied across different OxRRAM cells in a crossbar. This is illustrated in Figure 6a, where the minimum and maximum voltages are 0.4 and 0.57 V, respectively. Such voltage differences cause variation of the state transition time [see (2) and (3)]. Figure 6b shows such variation for OxRRAM cells in the crossbar, with each cell programmed to the HRS state. The minimum and maximum state transition times are 5,227 and 31,214 ms, respectively.

In a recent work, we have shown that the unit parasitic resistance of bitlines/wordlines increases from 1 Ω at 65 nm to 3.8 Ω at 16 nm [10]. Such increase in the value of parasitic resistance leads to a higher voltage applied across each OxRRAM cell in the crossbar, which further reduces its state transition time. To illustrate this, Figure 7 shows the variation in state transition time of OxRRAM cells in a crossbar at different process technology nodes. We make the following two key observations. First, the state transition time decreases with technology scaling. This is due to an increase in the voltage within the crossbar at scaled nodes. Second, the variation of state transition time increases at smaller nodes due to higher voltage and current variations [10].

Finally, the state transition time of OxRRAM cells also depends on the resistance state. Figure 8 shows the dependence of the minimum state transition time of OxRRAM cells in a crossbar for the four process technology nodes. We make two key observations. First, the state transition time reduces with technology scaling, which we have analyzed before. Second, the state transition time of an OxRRAM cell is higher when the cell is programmed in the HRS state for all process technology nodes. This is because the vertical filament growth phenomena (in the HRS state) in OxRRAM technology is slower than the lateral filament growth (in the LRS state).

During each inference operation, OxRRAM cells of a crossbar propagate spikes from a machine learning workload. To compute the *inference lifetime* of an OxRRAM cell, which is defined as the number of inference operations, it takes for the resistance state of the cell to drift from its programmed value, we let η be the average number of spikes through the cell per inference operation. Formally, inference lifetime $\mathcal{L}$ is

$$\mathcal{L} = \frac{t(\text{LRS}/\text{HRS})}{\eta}. \tag{4}$$

To ensure integrity of machine learning, that is, to prevent accuracy drop, the OxRRAM cell must be reprogrammed to the original resistance state once every inference lifetime. Since different OxRRAM cells in a neuromorphic crossbar have different inference lifetime, the reprogramming interval *t*RPI of model parameters to the hardware [see (1)] is defined as the minimum inference lifetime of all OxRRAM cells in the crossbar, that is,

$$t\text{RPI} = \min_{\forall i,j} \mathcal{L}_{i,j}. \tag{5}$$

The number of spikes propagating through an OxRRAM cell depends on the machine learning workload and how the workload is mapped to the crossbar. This is described next.

Workload dependency of inference lifetime

To understand the workload dependency of inference lifetime, we focus on (9). Here η is the average spikes per image through an OxRRAM cell implementing the machine learning workload. This is computed as follows. Consider our machine learning model is represented as $\mathcal{M}(N, S)$ with the set N of neurons and the set S of synapses. If x_i is the number of spikes communicated from a presynaptic neuron to a postsynaptic neuron via s_i, then the total number of spikes for the image is $\sum_i x_i$. In our implementation, each synapse (i.e., its weight) is programmed on an individual OxRRAM cell. Therefore, the number of spikes through all OxRRAM cells of the hardware is $\sum_i x_i$. We compute the average number of spikes per image through an individual OxRRAM cell as the sum of spikes for all images inferred by the model averaged over the number of images and synapses, that is,

$$\text{Avg. Spike Per Image} = \frac{\sum_{j=1}^{I} \sum_i x_i}{I \times |S|} \tag{6}$$

where I is the number of images inferred by $\mathcal{M}$.

Figure 9 shows the histogram of average spikes per image propagating through the synapses of VGGNet. We collected these statistics by analyzing CIFAR-10 training and test data sets. We see that there are 20 synapses in the model that communicate between 1 and 2 spikes per image, 30 synapses that communicate between 2 and 3 spikes per image, and so on. Therefore, some synapses propagate more spikes than others.

If we consider two different synapses of a model with different spike count, then the one with a higher number of spikes will result in a lower inference lifetime when mapped to the OxRRAM cell at a specific position in the crossbar.

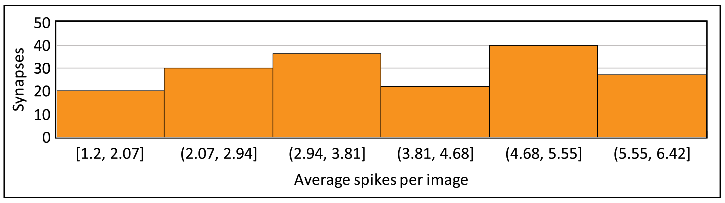


Figure 9. Spike distribution across the synapses of VGGNet.

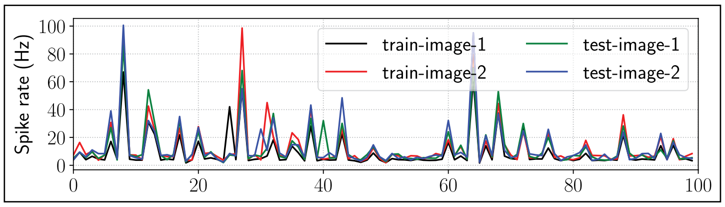


Figure 10. Spike rate of 100 randomly selected neurons in VGGNet for two training images and two test images.

Additionally, the spike count on a synapse also depends on the input presented to a model. To illustrate this, Figure 10 plots the spike firing rate of 100 randomly selected neurons in VGGNet. We observe that the spike firing rate of a neuron in VGGNet depends on the image presented to the model.

The proposed design methodology incorporates such application and model-dependent behavior to better optimize the synapse mapping to OxRRAM cells. This is described next.[3]

Proposed design methodology

Figure 11 shows the proposed system architecture designed in NeuroXplorer [6]. A machine learning model is first trained using training data. The model parameters are stored in memory. The trained model is clustered using the graph partitioning algorithm of NeuroXplorer. For each cluster, an optimization is performed to map the neurons and synapses of the cluster to the OxRRAM cells of a crossbar, by exploiting: 1) spike data collected from the training set and 2) technology-specific state-transition time data obtained from characterizing the hardware. The cluster optimization step generates the parameter reprogramming interval *t*RPI, which is then used to periodically reprogram the model parameters to the

[3] The scope of the current work is on design-time approaches in mitigating resistance drift. Our future work will involve designing a run-time framework to evaluate spike count of synapses based on the model input and enable remapping of the synaptic connections to further reduce the system overhead.

hardware via bandwidth-limited memory channels. The new blocks that we introduce in NeuroXplorer are shown in red in Figure 11.

We introduce the following notations to formulate the cluster optimization problem.

$\mathcal{M}$ = Set of presynaptic neurons of a cluster.

$\mathcal{N}$ = Set of postsynaptic neurons of a cluster.

$\mathcal{S}$ = Set of synapses of a cluster .

$\eta_{i,k}$ = Spikes on the synapse $s_{i,k} \in \mathcal{S}$.

$e_{j,l}$ = State transition time of the (j, l)th OxRRAM cell in a crossbar.

$$x_{i,j} = \begin{cases} 1, & \text{if presynaptic neuron } m_i \in \mathcal{M} \text{ is mapped to crossbar input } I_j \\ 0, & \text{otherwise.} \end{cases}$$

$$y_{k,l} = \begin{cases} 1, & \text{if postsynaptic neuron } n_k \in \mathcal{N} \text{ is mapped to crossbar output } O_l \\ 0, & \text{otherwise.} \end{cases}$$

Following are the constraints.

- A presynaptic neuron can be mapped to exactly one input port of a crossbar, that is,

$$\sum_{\forall j} x_{i,j} = 1 \quad \forall i. \tag{7}$$

- A postsynaptic neuron can be mapped to exactly one output port of a crossbar, that is,

$$\sum_{\forall l} y_{k,l} = 1 \quad \forall k. \tag{8}$$

We formulate the optimization problem as follows. $x_{i,j} \cdot y_{k,l}$ defines the mapping of synapse $s_{i,k} \in \mathcal{S}$ to the (j, l)th OxRRAM cell in the crossbar. The inference lifetime of this mapping is

$$\mathcal{L}_{i,j,k,l} = \frac{e_{j,l}}{\eta_{i,k}}. \tag{9}$$

The optimization problem is

$$\textbf{Maximize } t\text{RPI} = \min_{\forall i,j,k,l} x_{i,j} \cdot y_{k,l} \cdot \mathcal{L}_{i,j,k,l}. \tag{10}$$

The nonlinear operation of multiplication of two binary variables $x_{i,j}$, and $y_{k,l}$ is linearized by introducing a new product variable $z_{i,j,k,l}$, with the following additional constraints.

- If $x_{i,j} = 0$ and/or $y_{k,l} = 0$, then $z_{i,j,k,l} = 0$, that is,

$$z_{i,j,k,l} \le x_{i,j} \text{ and } z_{i,j,k,l} \le y_{k,l}. \tag{11}$$

- If $x_{i,j} = 1$ and $y_{k,l} = 1$, then $z_{i,j,k,l} = 1$, that is,

$$z_{i,j,k,l} \ge x_{i,j} + z_{i,j,k,l} - 1. \tag{12}$$

The new optimization problem is

$$\textbf{Maximize } \min_{\forall i,j,k,l} z_{i,j,k,l} \cdot \mathcal{L}_{i,j,k,l}. \tag{13}$$

This max–min optimization problem is a convex one [proof of Karush–Kuhn–Tucker (KKT) conditions are omitted for space limitations]. The problem can be solved using CVXPY by introducing a slack variable τ as

$$\textbf{Maximize } \tau \ni \tau \le z_{i,j,k,l} \cdot \mathcal{L}_{i,j,k,l} \quad \forall i,j,k,l. \tag{14}$$

To incorporate the criticality of a synaptic connection, we assign a very small number as the spike count η for the synapse. In other words, the spike count of critical synapses is identified using the training set, while those for noncritical synapses are set to a very small value. In this way, we force $\mathcal{L}$ [see (9)] to a very large value for the noncritical synapses. This allows the convex optimizer to eliminate them from the critical path of determining the reprogramming interval tRPI [see (14)].

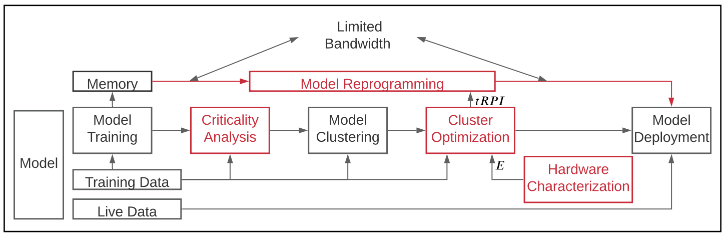


Figure 11. System architecture.

Table 1. Major simulation parameters extracted from [2].

Neuron technology	16nm CMOS (original design is at 14nm FinFET)
Synapse technology	HfO_2-based OxRRAM [2]
Supply voltage	1.0V
Energy per spike	23.6pJ at 30Hz spike frequency
Energy per routing	3pJ
Switch bandwidth	3.44 G. Events/s

Results and discussion

We evaluate the proposed design methodology for OxRRAM-based neuromorphic hardware. We configure NeuroXplorer with the hardware parameters listed in Table 1.

We use five commonly used CNN applications with 2-bit quantized synaptic weights. These applications are described in Table 2.

We evaluate the following techniques.

- *SpiNeMap:* This baseline approach first clusters a machine-learning inference model to minimize the intercluster spike communication [12]. Clusters are then mapped to crossbars of a neuromorphic hardware with synapses of each cluster implemented randomly on OxRRAM cells of a crossbar.
- *Endurer:* This is our previous work, which addresses the reprogramming of model parameters on crossbars of a neuromorphic hardware to maintain model integrity [5]. A machine learning model is clustered using SpiNeMap. Clusters are placed to crossbars to maximize inference lifetime. To map cluster synapses to the OxRRAM cells of a crossbar, Endurer uses a binary nonlinear optimization problem formulation.
- *Proposed:* The proposed approach is based on Endurer. It introduces the following two new changes to Endurer: 1) it characterizes a machine learning model to identify noncritical synapses such that they could be eliminated from the critical path of determining the reprogramming interval and 2) the convex optimization formulation and the proposed linearization technique improves the solution quality and improves the speed-up, accelerating the design space exploration.

Accuracy

Figure 12 reports the accuracy improvement due to periodic reprogramming of model parameters in Endurer and the proposed approach compared to SpiNeMap, where no reprogramming is performed. We observe that by enabling reprogramming of model parameters, model accuracy can be improved by 25% (between 3% and 87%). This is because, without periodic reprogramming in place, model parameters may drift due to frequent accesses of OxRRAM cells where these parameters are programmed. Parameter drift leads to lower accuracy. Additionally, the extent of accuracy impact depends on the specific model that is programmed to the hardware. For AlexNet, we observe a 47% drop, while for ResNet and DenseNet, the drop is only 4%.

System overhead

Figure 13 reports the system overhead of the proposed approach compared to Endurer for the evaluated CNNs. Results are normalized to Endurer. Since SpiNeMap does not involve periodic reprogramming, there is no system overhead. We have therefore not shown SpiNeMap in the figure.

We observe that the system overhead of the proposed approach is on average 35% lower than Endurer (between 23% and 50%). This improvement is due to the increase of reprogramming interval in the proposed approach. Such improvement is attributed to two factors. First, noncritical synapses are not on the critical path for determining the reprogramming interval *t*RPI in the proposed approach, while such synapses are factored in determining *t*RPI in Endurer. Second, the convex optimizer CVXPY of the proposed approach generates a better solution than the approximate binary nonlinear optimization technique of Endurer.

We also observe that the improvement is usually higher for models with higher fraction of noncritical synapses. Therefore, the improvement for ResNet is higher than LeNet.

In this work, we study the resistance drift-related reliability issues in OxRRAM-based neuromorphic hardware used to implement machine learning inference models. Through circuit-level simulations,

Table 2. CNN applications used to evaluate the proposed design.

CNN	Dataset	Neurons	Synapses	Avg. Spikes/Image	Accuracy-Full	Accuracy-2 bit
LeNet	CIFAR-10	80,271	275,110	724,565	86.3%	55.4%
AlexNet	CIFAR-10	127,894	3,873,222	7,055,109	66.4%	69.5%
VGGNet	CIFAR-10	448,484	22,215,209	12,826,673	81.4 %	55.3%
ResNet	CIFAR-10	266,799	5,391,616	7,339,322	57.4%	48.0%
DenseNet	CIFAR-10	365,200	11,198,470	1,250,976	46.3%	28.2%

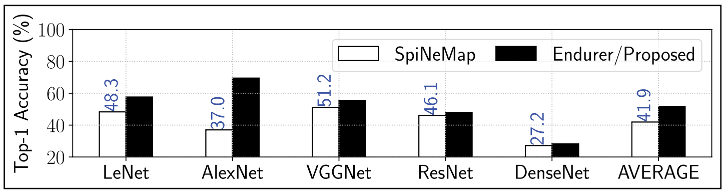


Figure 12. Accuracy improvement due to periodic reprogramming.

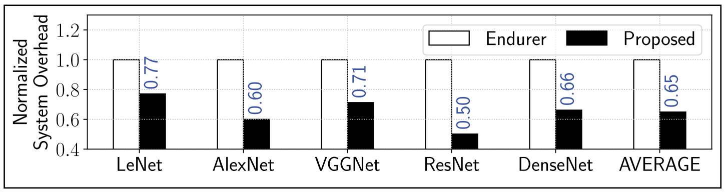


Figure 13. Overhead improvement of the proposed approach.

we show the dependence of these issues on: 1) the resistance state of an OxRRAM cell (model parameter dependency); 2) the current through the cell (circuit dependency); and 3) the spikes propagating through the cell (workload dependency). We incorporate this study in a system software framework and show a significant accuracy drop due to resistance drift. To maintain the integrity of machine learning inference, model parameters need to be reprogrammed to the hardware periodically, which incurs a significant system overhead. We propose an approach to minimizing this system overhead by first analyzing a machine learning model to identify noncritical synapses and then proposing a convex optimization solution to maximize the reprogram interval. The proposed optimizer eliminates the noncritical synapses from the critical path of determining the reprogram interval. Evaluations with five commonly used CNN applications show an average 35% improvement in the system overhead. ■

Acknowledgments

This work was supported by the National Science Foundation Faculty Early Career Development Award under Grant CCF-1942697 (CAREER: Facilitating Dependable Neuromorphic Computing: Vision, Architecture, and Impact on Programmability).

References

[1] M. L. Varshika et al., “Design of many-core big little μBrain for energy-efficient embedded neuromorphic computing,” in *Proc. DATE*, 2022, pp. 1–7.

[2] A. Mallik et al., “Design-technology co-optimization for OxRRAM-based synaptic processing unit,” in *Proc. VLSIT*, 2017, pp. T178–T179.

[3] W. Shim et al., “Impact of read disturb on multilevel RRAM based inference engine: Experiments and model prediction,” in *Proc. IRPS*, 2020, pp. 1–5.

[4] V. Milo et al., “Multilevel HfO_2-based RRAM devices for low-power neuromorphic networks,” *APL Mater.*, vol. 7, no. 8, 2019, Art. no. 081120.

[5] S. Song et al., “Improving inference lifetime of neuromorphic systems via intelligent synapse mapping,” in *Proc. ASAP*, 2021, pp. 17–24.

[6] A. Balaji et al., “NeuroXplorer 1.0: An extensible framework for architectural exploration with spiking neural networks,” in *Proc. ICONS*, 2021, pp. 1–9.

[7] S. Han et al., “Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding,” 2015, *arXiv:1510.00149*.

[8] A. Chaudhuri et al., “Hardware fault tolerance for binary RRAM crossbars,” in *Proc. ITC*, 2019, pp. 1–10.

[9] T. Spyrou et al., “Neuron fault tolerance in spiking neural networks,” in *Proc. DATE*, 2021, pp. 743–748.

[10] T. Titirsha et al., “Endurance-aware mapping of spiking neural networks to neuromorphic hardware,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 2, pp. 288–301, Feb. 2022.

[11] A. Paul et al., “Design technology co-optimization for neuromorphic computing,” in *Proc. IGSC Workshops*, 2021, pp. 1–6.

[12] A. Balaji et al., “Mapping spiking neural networks to neuromorphic hardware,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 1, pp. 76–86, Jan. 2020.

Ankita Paul is pursuing a PhD with Drexel University, Philadelphia, PA 19147 USA, under the supervision of Dr. Anup Das. Her research interests include brain inspired computing, deep learning, and machine learning. Paul has a bachelor’s degree from the West Bengal University of Technology, Kolkata, India.

Shihao Song is pursuing a PhD with Drexel University, Philadelphia, PA 19147 USA, under the supervision of Dr. Anup Das. His research interests include computer architecture, non-volatile memory, and compiler design for neuromorphic hardware and accelerators. Song has a bachelor’s from Drexel University.

Twisha Titirsha is pursuing a PhD with the Department of Electrical and Computer Engineering, Drexel University, Philadelphia, PA 19147 USA. Her research interests include computer architecture, non-volatile memory, and mixed-signal circuit design. Titirsha has a bachelor’s from the Military Institute of Science and Technology, Dhaka, Bangladesh.

Anup Das is an assistant professor at Drexel University, Philadelphia, PA 19147 USA. His research focuses on neuromorphic computing and architectural exploration. Das has a PhD in embedded systems from the National University of Singapore, Singapore. He is a Senior Member of IEEE.

■ Direct questions and comments about this article to Anup Das, Department of Electrical and Computer Engineering, Drexel University, Philadelphia, PA 19147 USA; anup.das@drexel.edu.

Using STLs for Effective In-Field Test of GPUs

Josie E. Rodriguez Condia
Politecnico di Torino
10129 Turin, Italy

Felipe Augusto da Silva
Cadence Design Systems
85622 Munich, Germany
Delft University of Technology
2028 Delft, The Netherlands

Ahmet Çağrı Bağbaba
Cadence Design Systems
85622 Munich, Germany
Tallinn University of Technology
19086 Tallinn, Estonia

Juan-David Guerrero-Balaguera
Politecnico di Torino
10129 Turin, Italy

Said Hamdioui
Delft University of Technology
2028 Delft, The Netherlands

Christian Sauer
Cadence Design Systems
85622 Munich, Germany

Matteo Sonza Reorda
Politecnico di Torino
10129 Turin, Italy

Editor's notes:
GPUs have seen an increased adoption in autonomous systems. This article assesses the fault coverage that can be attained through software self-test strategies for in-field test of GPUs.
—Nicola Nicolici, McMaster University

■ **MODERN GRAPHICS PROCESSING** units (GPUs) are manufactured using cutting-edge technologies but are prone to suffer from in-field errors and reliability issues [1]. The flexibility and computational power of GPUs push their adoption in developing advanced driver-assistance systems (ADASs) and sensor fusion solutions in the automotive and autonomous systems domains. However, the premature aging and wear-out features in new transistor technologies promote the rising of permanent faults during the in-field operation. In safety-critical applications, unaffordable failures caused by faults can induce the entire system to fail or even result in catastrophic consequences if no appropriate measures are taken promptly. Hence, the development of countermeasures for the in-field detection of faults is of great importance in GPUs.

Digital Object Identifier 10.1109/MDAT.2022.3188573
Date of publication: 5 July 2022; date of current version: 10 March 2023.

Published works, addressing in-field fault detection for GPUs, can be classified into three classes: 1) design for testability (DfT) methods, which are purely hardware-oriented; 2) hybrid approaches, which combine hardware structures with reconfigurable capabilities at the software level; and 3) software-based self-test (SBST) solutions. DfT schemes are widely used for the end-of-production test in current devices. However, they are not always available for in-field operation and may not satisfy time constraints in many applications. Furthermore, hybrid solutions, based on the addition or use of available structures (i.e., performance counters) to extend the fault observability of a module, must be included in the design phases by modifying the hardware–software interface to provide instruction-based control of the included structures. Jagannadha et al. [2] proposed an in-system-test architecture based on the combination of DfT schemes and hybrid structures to detect faults and provide diagnosis features during the in-field operation of system-on-chips (SoCs) and GPUs. However, a massive effort is required to develop and integrate a coordinated ecosystem to design and verify the device. On the other hand, the SBST strategy is a noninvasive and flexible approach to perform functional in-field tests of processor-based systems, which has been widely adopted in processor testing [3]. Nowadays, semiconductor companies and IP providers give SBST support for their safety-critical products (e.g., automotive). In detail, the SBST strategy resorts to specially written software-test libraries (STLs) composed of suitably developed test programs (TPs) able to achieve a given structural fault coverage (FC) when run by the CPU with limited or null external support. A TP is a suitable sequence of selected instructions applying test patterns to a given unit and propagating fault effects up to some observation points. These are typically developed starting from high-level abstractions of a design (RT-level) and then progressively reaching and refined at lower levels (Gate-level). Moreover, TPs can often be split into small chunks of code fitting in the idle times of an application and thus more easily matching time constraints. In the past, numerous works developed effective STLs for CPUs. However, only a few works used SBST strategies for in-field tests in GPUs. Clearly, some of the techniques used for CPUs can be extended to GPUs as well. Nevertheless, GPUs have some specific features and characteristics (e.g., implicit parallelism, parallel scheduling, and shared memory management), which demand special strategies to test the corresponding hardware modules. Di Carlo et al. [4] adopted several processor-based techniques into TPs for the execution units, register files, and main memories in a GPU. Nevertheless, observability issues restricted the assessment of the FC. Another work [5] addressed the test of control units (scheduling controller). However, the development of customized approaches was required. In conclusion, prior works on in-field tests are unaffordable due to huge complexity and intrusiveness (DfT and hybrid cases) or suffer from generality (SBST case), making them not fully suitable for GPUs. Hence, there is a need of providing a complete solution for in-field tests.

This work, for the first time, evaluates the overall effectiveness of employing the SBST strategy for the in-field test of all logic modules of a GPU core. Moreover, this work experimentally quantifies the FC achievable on the logic modules in a GPU core. Finally, it evaluates how suitable STLs can support the failure modes and effects analysis (FMEA) required in all safety-critical domains. The main contributions of this work can be summarized as follows.

- A general overview of the characteristics and strategies to develop STLs for GPUs.
- An evaluation (the first publicly available, to the best of our knowledge) of the overall FC obtained on a GPU core with the STL execution.
- A report about the failure modes effects and diagnostic analysis (FMEDA) process on a GPU core using STLs as the only fault-tolerance mechanism.

This work resorts to the FlexGripPlus model, describing one low-level microarchitecture of NVIDIA, to evaluate and validate the development of STLs for GPUs. The experimental results show that up to 92.6% of the stuck-at faults (SAFs) in the logic blocks of a GPU core can be covered using the STLs we developed. The FMEDA analysis shows that these results enable to qualify the considered modules inside a GPU core via STLs at least for the ASIL B level. Higher levels can be achieved by combining the STLs with other safety mechanisms.

Architectural organization of GPUs

General overview

This section employs NVIDIA's terminology to describe the architectural organization of a GPU.

GPUs are special-purpose processors organized as arrays of parallel cores [streaming multiprocessors (SMs)]. Each SM adopts the single-instruction multiple-data (SIMD) paradigm or variations, such as single-instruction multiple-thread (SIMT) by NVIDIA. Internally, each SM comprises several pipeline stages and uses a specific instruction set architecture partly resembling RISC ones with extensions to support parallelism.

A host controller (CPU) submits a parallel program to the GPU for processing. Then, the program is distributed among the available SMs by the schedulers. Internally, the scheduler controllers manage and trace the operation of a group of threads (*warp*), which are operated in parallel on individual execution units [scalar/streaming processors (SPs)]. Each SP is composed of an integer (INT) and a floating-point core (FP). Moreover, the SM includes other hardware accelerators (SFUs) as well.

Each SM has access to several levels of the memory hierarchy (register file, shared, local, constant, and main memory). The register file and the shared

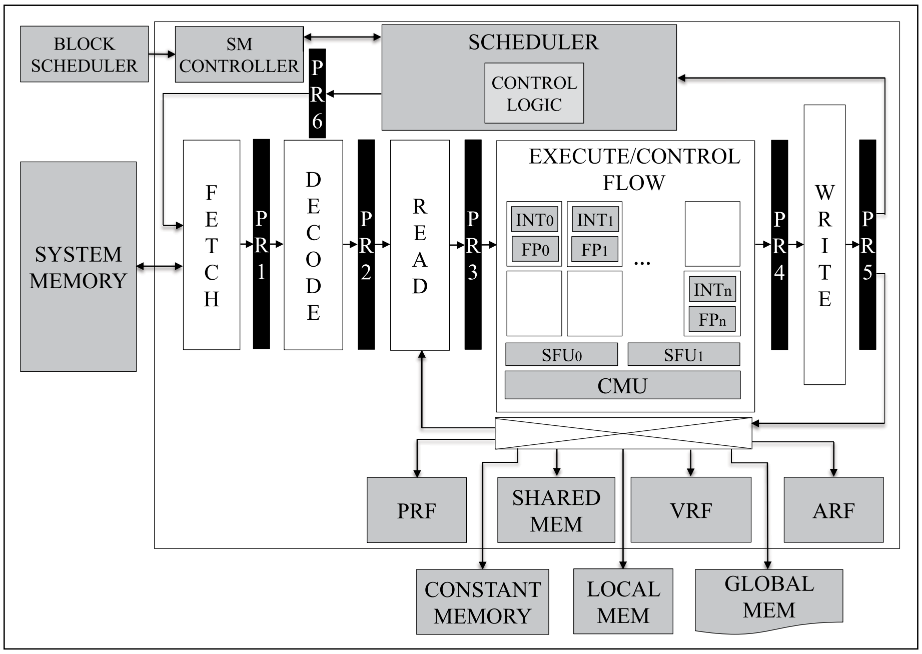


Figure 1. General scheme of an SM in FlexGripPlus.

memory are organized in banks for parallel access and store the individual and shared operands and results for each thread, respectively. Both resources and the first levels of cache are located inside the SM. The second-level caches, the constant, and the main memories are located outside as a shared resource among the SMs.

FlexGripPlus model

FlexGripPlus is an open-source soft-GPU model based on the NVIDIA G80 microarchitecture and fully described in VHDL [6]. FlexGripPlus is compatible with the CUDA programming environment (SM 1.0) and is based on a set of SMs supporting up to 52 assembly (SASS) instructions.

Each SM is divided into five pipeline stages (*Fetch*, *Decode*, *Read*, *Execute/Control-Flow*, and *Write*), as shown in Figure 1. The number of SPs in the *Execute* stage is configurable among 8, 16, or 32. Moreover, pipeline registers ("PRx") are located between the pipeline's stages. Each SM also includes three register files (*Vector Register File* "VRF"), (*Address Register File* "ARF"), and (*Predicate Register File* "PRF"), devoted to storing operands, addresses, and predicate flags of each thread, respectively.

Each SM includes one scheduler controller and a divergence management unit (DMU) for intra-warp divergence control and execution.

In general, the FlexGripPlus model holds the same basic functional modules of a commercial GPU, including scheduler controllers, parallel execution units, file registers, and pipeline stages. Nevertheless, the current memory hierarchy in FlexGripPlus differs from the included in commercial devices by missing the cache memories.

Despite the few structural limitations, the FlexGripPlus model includes a low-level detailed microarchitectural description of an NVIDIA GPU and is employed as a tool to evaluate the effectiveness of STLs for GPUs developed using the SBST strategy.

SBST strategies for GPUs

STLs developed with the SBST strategy can be deployed as complementary mechanisms to monitor the status of a GPU during its operative life and contribute to identifying possible fault effects. In fact, the main advantage of STLs is the ability to detect faults with zero hardware costs. Moreover, STLs test a device at the operational speed and normal conditions, thus also addressing delay faults and avoiding overtesting.

In the functional-safety domain, the identification and management of faults in a device are mandatory. Some faults can be classified as *safe,* when they are proved not to be able to produce any failure in the considered operational scenario. Safe faults are not considered when computing the achieved FC.

In this domain, STLs can be used as safety mechanisms and increase reliability by guaranteeing the in-field detection of a sufficient percentage of faults, thus matching the requirements of the functional safety standards, possibly in combination with other mechanisms (e.g., ECC for memories, and watchdogs). STLs are widely used for CPUs but they can also be adopted for accelerators, such as GPUs, which demand periodic testing solutions when used in safety-critical applications. In this case, we must consider two main features: 1) most in-field faults in GPUs can only be observed by looking at results they produce in memory (as the main observation point) and 2) the development of TPs requires architectural details from a targeted unit. In general, any TP is mainly executed following four steps: 1) initialization; 2) test pattern's injection; 3) fault effect's propagation to any observation point; and 4) identification (see Figure 2). In the execution of a TP, several loops can apply different test patterns or propagate their effects. However, TPs for GPUs must face the addressing of each module exploiting the implicit parallelism and operational constraints (e.g., divergence and thread-synchronization). For this purpose, these TPs must exploit three main characteristics of the parallel operation of GPUs.

- Instruction parallelism.
- Distributed scheduling.
- Management of functional units and memory resources.

The following sections summarize some specific strategies and algorithms used in the development of TPs for STLs targeting GPUs. It should be noted that each GPU module may require a combination of different approaches. Fortunately, one TP may target the test of several modules in parallel.

Extending functional test techniques from CPUs to GPUs

Two approaches originally developed for CPUs can be adopted and extended to the GPU domain: automated and deterministic [3], [7].

On the one hand, the automated approaches comprise pseudorandom- and ATPG-based methods. The first method focuses on TPs based on a group of instructions randomly selected in combination with pseudorandom operand values. This method can exploit evolutionary algorithms to select the most suitable instructions and operands for a TP. The second method resort to Automatic Test Pattern Generation (ATPG) tools to analyze and extract test patterns from a hardware module. Then, these patterns are translated into equivalent instructions, so composing one or more TPs. However, it is possible that some test patterns cannot be translated and must be ignored (possibly resulting in safe faults). In both cases, several iterations are used in the development of each TP to improve its correct operation and reduce unnecessary overhead costs for the in-field operation.

In any case, TPs using either automated or deterministic approaches must include three strategies: 1) parallel pattern management (PPM); 2) *signatures per thread* (or SpT) [9]; and 3) parallel injection.

The first strategy (PPM) organizes and aligns similar test patterns and expected results as consecutive memory operands, so optimizing the performance in memory management and exploiting possible execution loops. Then, each thread in the TP can address individual or shared test patterns from memory.

The SpT mechanism is based on the computation, within each thread in a TP, of a signature providing fine-grain fault-observability out of the values produced by the target module during its operation, thus propagating fault effects as errors on the computing signature and allowing fault detection. Each SpT is described and computed in software by mimicking a multiple-input shift register or a counter, which reduces the number of instructions per TP while providing extended observability. In the end, each SpT is stored in memory. The GPU itself (or the host) checks for the presence of faults by comparing a produced signature with the expected one, which is precalculated by the TP itself (in the development and verification phases) with minimal performance overhead (<5%) and finally stored in specific memory regions available during the operation of the TPs. Those precalculated golden signatures avoid latencies at the in-field operation of TPs.

The parallel injection techniques take advantage of thread parallelism in *warps* or *blocks* to excite a module with different test patterns (one per thread), thus exploiting parallelism to increase the operational performance of a TP, which is effective in either individual unit or regular structures.

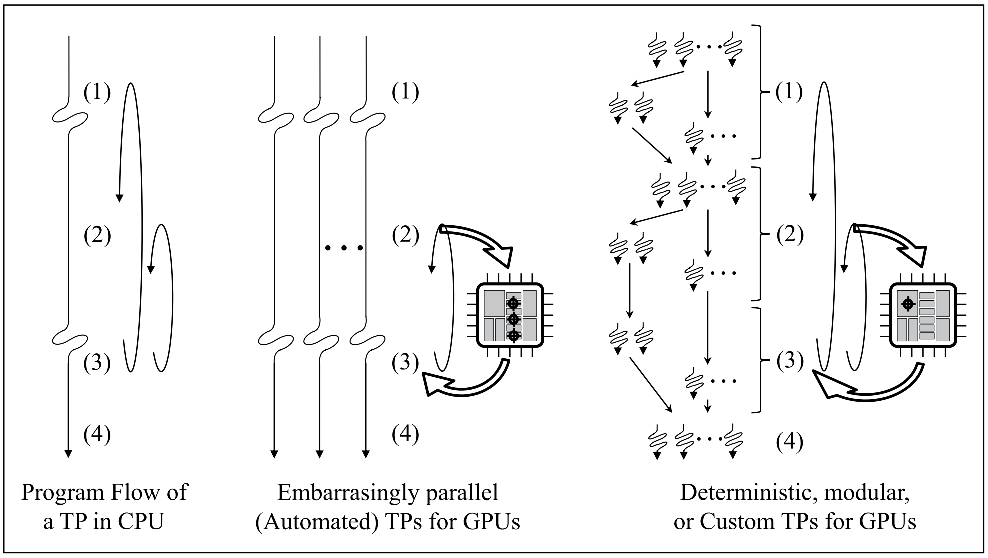

(1)
(2)
(3)
(4)
Program Flow of a TP in CPU
Embarrasingly parallel (Automated) TPs for GPUs
Deterministic, modular, or Custom TPs for GPUs

Figure 2. General scheme of the execution flow of TPs for CPUs and GPUs.

Pseudorandom and ATPG-based approaches are effective in regular structures of a GPU, such as the functional units and the register file, since these structures are addressed (and tested) in parallel. Moreover, the static organization and the understanding of distribution policies in the schedulers allow the development of embarrassingly parallel TPs (see Figure 2), exploiting the multithread parallelism to inject patterns and also reducing the in-field execution of TPs [8]. On the other hand, deterministic approaches exploit the functionality and structure in a module to deploy well-defined algorithms, such as March algorithms for internal memories (e.g., within the controllers) [4].

It must be noted that, when using a deterministic approach, the adaptation of a method may require additional steps (i.e., initialization and propagation patterns) to face the parallel operational constraints in a GPU, but additional efforts are required to control intrawarp divergences, thread synchronizations, and concurrent loops when injecting test patterns, as depicted in Figure 2.

Multikernel approach

TPs in this approach utilize a divide-and-conquer strategy to target special modules commonly managing configuration parameters devoted to controlling and configuring the parallel operation in a GPU. These configuration parameters (i.e., memory addresses, number of threads, blocks, grids, and registers per thread) come from the program and configure modules (i.e., the constant memory and the schedulers) for the operative interval of the program.

In this case, multiple TPs (*kernels*) employ the policies of scheduling and the resource's management to target a different set of faults inside a module. More in detail, each TP uses different combinations of configuration parameters, which also serve as indirect test patterns, so activating different regions (and exciting possible faults) in a module. The multikernel approach is effective when testing modules managing parallel parameters, such as the scheduling controllers and the pipeline registers. Further details can be found in [9]. Finally, this approach can be extended to other modules with similar fault activation and propagation restrictions (i.e., global schedulers outside the SM).

Modular kernel approach

This approach exploits a top-down strategy to develop modular routines to build TPs for complex units in a GPU. The modular description of a routine starts from a high abstraction level and is then customized. In this approach, the most suitable instructions to activate and propagate faults inside the target unit are selected considering two factors: 1) the parallel observability and controllability features and 2) the architectural description and operational constraints of a target unit.

First, the controllability and observability features are determined for a target unit. In this case, suitable instructions (i.e., "Push" and "Pop" in a stack) are used to provide both features as initial conditions in a TP. Then, several routines to inject test patterns while exploiting parallelism are designed (in CUDA or SASS) and linked, considering the operational constraints of the unit. These routines are the basic components of a TP and describe the operation of any thread. The flexibility of the approach allows the development and exploration of several parallel routines providing the same functionality. Moreover, the execution flow in a TP can be adopted according to the selected routines. Finally, the routines are integrated as a single TP and refined for performance or FC.

The modular approach is effectively applied to complex units in a GPU, such as the DMU and the embedded memories [10].

Customs approaches

The custom approaches require the manual development of TPs following some specific algorithm that takes into account the architecture of the units, their functional operation, the expected behavior, their restrictions, and the target fault model. These TPs target particular modules in the GPU, which do not exist in CPUs (such as the scheduler controllers [5] and the special-purpose memories [4]). In detail, the TPs are based on algorithms causing controlled divergence, the combination of sequences of embarrassingly parallel, and serial-thread executions on a set of threads to excite and propagate fault effects.

This approach requires a deep knowledge of the GPU's low-level microarchitectural details, their parallel operations and the use of parallelism, distributed scheduling, and available hardware resources to provide specific test solutions per module.

Setup and preliminary GPU analysis

The evaluation of the STLs (developed using all techniques described in the previous section) resorted to a commercial fault simulation environment targeting the units in the FlexGripPlus model. This framework uses the RT-level description of the GPU and evaluates each TP by injecting SAFs into the logic of every logic module.

In the experiments, we targeted the evaluation of all logic modules and embedded memories in the GPU core. One fault is detected when at least one mismatch is found after comparing results from a golden execution and a faulty one. The flexibility of the tool allows the selection of the memory buses and the output control signals as the main in-field observation points of faults. It is worth noting that the main memory and the memory controllers were not targeted, since these are not part of the GPU core.

Three preliminary architectural analyses identified safe faults in the GPU. The first analysis is based on the architectural propagation analysis, which consists of evaluating the propagation paths of each fault in the design up to the observability points. In addition, the fault activation analysis evaluates the inputs of the GPU and identifies those faults that cannot be excited. Finally, a barrier analysis provides the analysis of the structural and operational effects of removing modules in the GPU [11].

Table 1 reports the number of identified safe faults in the FlexGripPlus SM. Interestingly, the fault propagation analysis effectively identified most of the safe faults per module (>90%). The other two methods effectively identified faults in the GPU's special structures, such as inside the scheduler. A postchecking process was required to determine any detectable and dangerous fault (i.e., faults in locations that remain temporarily fixed by the effect of the kernel or host configuration, but in other conditions, these may cause misbehaviors), so removing them from the list of safe faults.

STL effectiveness evaluation

In the experiments, FlexGripPlus was configured with one SM and 32 SPs. A set of 18 TPs were implemented using the methods outlined in the previous section. Each TP is developed according to the unit's features.

Three automatic TPs targeted the functional units and the decode unit by exploiting instructions that

Table 1. Untestable fault identification.

Module	Faults	Safe faults (%)	Remaining faults
SMP controller	2,508	20.26	2,000
Warp Unit	18,804	23.44	14,396
Pipeline Fetch	737	17.77	606
Pipeline Decode	1,514	21.53	1,188
Pipeline Execute	142,124	23.88	108,185
Address Register File	32,800	0.10	32,768
Vector Register File	131,750	0.51	131,072
Predicate Register File	33,038	0.82	32,768
Divergence Stack Memory	4,568	7.53	4,224
Overall GPU's SM core	367,843	11.05	327,207

excite as many patterns (operands) as threads on them. Moreover, three deterministic and modular TPs targeted the embedded memories, using the operational features (writing and reading methods) to excite the units. Nine TPs used the multikernel approach targeting programmable pipeline registers. Finally, three custom TPs focus on exciting controllers and dispatchers in the GPU.

In the end, 15 fault injection campaigns were performed on the complete GPU model, after generating the full list of SAFs, safe faults were first removed. Moreover, in the fault campaigns, the total number of faults (327,207) was reduced by injecting faults only in one module among the regular modules in the GPU (i.e., one SP and the associated register file per core, instead of the 32 execution units). As a result, in each fault simulation campaign, we injected 141,140 SAFs.

Since the current version of FlexGripPlus does not include accurate descriptions of the caches, the memory controllers were not addressed.

Table 2 reports the obtained FC figures. As observed, the developed STLs mainly focused on the largest modules in the GPU's core architecture, such as the execution units, internal logic, and embedded memories, which account for more than 90% of faults in an SM. Although some TPs provide moderate fault detection in some modules of the GPU (e.g., controllers), the small size of these structures does not significantly affect the overall FC in the GPU core. Further efforts could be made to develop suitable TPs specifically addressing these modules.

Previous results demonstrate that STLs can be effectively developed and provide a high FC. Although the obtained results were focused on one GPU core, the implemented TPs are scalable and results can be extended to multi-SM GPUs. Furthermore, the development of STLs can be applied to other GPU architectures.

Table 2. FC results per module.

Module	Detected faults	FC (%)
SMP controller	1,156	57.8
Warp Unit	8,416	58.5
Pipeline Fetch	538	88.8
Pipeline Decode	837	70.4
Pipeline Execute	91,429	84.5
Address Register File	32,768	100.0
Vector Register File	131,072	100.0
Predicate Register File	32,768	100.0
Divergence Stack Memory	4,139	98.0
Overall GPU's SM core	303,295	92.6

Functional-safety evaluation

The calculation of the FC is an indication of the design safety based on the efficiency of a given safety mechanism (SMech). However, it is not sufficient to assure compliance with functional safety standards, like ISO26262; for such a purpose, we need to determine the reduction in the probability of system failures, also known as the failure in time (FIT) rate. The single-point faults metric (SPFM), which represents permanent faults' potential to violate safety-related functionalities, is defined by ISO26262 as evidence of safety integrity [12]. The SPFM considers the total FIT rate (λ) and the contribution of the fault classes.

- Single-point faults (λSPF): Not covered by SMechs.
- Residual faults (λR): Undetected by SMechs.

The SPFM can be calculated according to

$$\mathrm{SPFM} = 1 - \frac{\sum(\lambda \mathrm{SPF} + \lambda R)}{\sum \lambda}. \quad (1)$$

The primary methodology for determining the safety metrics parameters is the FMEDA, which correlates IC components (*Gates*, *Flip-flops*, and *Memory cells*) to failure modes (FMs). Then, by computing λ of individual IC components, the FC, and the Safe faults, we can determine the total λ of each FM.

First, the FMs are defined and the design components mapped. For FlexGripPlus, we considered 28 subparts (components inside the GPU core, including local controllers, functional units, embedded memories, and registers). Each subpart was analyzed to determine function-specific FMs. After mapping each FM to the appropriate design component(s), we evaluate the percentage of Safe faults and the FC. The FlexGripPlus' FMEDA comprises 92 FMs mapped to 2,751,088 gates, 1,507,085 flops, and 784,224 memory cells.

The analysis of FlexGripPlus, considering the 15-nm FinFET-based Open Cell Library, resulted in a total λ of 10.08 FIT (based on IEC 62380 Electronic Reliability Prediction Standard), which defines a base FIT Rate for the components of a given tape-out technology. In this case, the unit's base FIT considers digital (NAND2 gate's area) and memory (cell's area) components. Then, we multiply the number of gates and cells, mapped to each FM, by the digital and memory FITs, respectively; from these, the implemented safety strategies provide the following results:

- Detected by the STL: 9.17 FIT.
- Undetected (λR): 0.57 FIT.
- Safe faults (λS): 0.33 FIT.

Finally, reducing λR by increasing λS and FC directly impacts the SPFM. The proposed Safety technique based on only STLs for FlexGripPlus resulted in an SPFM of 94.27%, allowing ASIL B assessment without hardware modifications to the logic units of an SM and without any other SMech.

This work is the first to provide a quantitative evaluation of the effectiveness of STLs for the in-field testing of GPU cores. The reported results showed that an SAF coverage of more than 92% could be obtained on the logic modules and embedded memories. The functional-safety results (SPFM of 94.27%) show the effectiveness of STLs as a safety mechanism for SMs in GPUs.

The results allow us to state that the SBST strategy can be used as an effective solution, possibly combined with other strategies, to guarantee the reliability and functional safety of GPU-based applications for safety-critical domains. ■

Acknowledgments

This work was supported by the European Commission through the Horizon 2020 RESCUE-ETN Project under Grant 722325.

References

[1] D. Tiwari et al., "Reliability lessons learned from GPU experience with the Titan supercomputer at oak ridge leadership computing facility," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2015, pp. 1–12.

[2] P. K. D. Jagannadha et al., "Special session: In-system-test (IST) architecture for NVIDIA drive-AGX platforms," in *Proc. IEEE 37th VLSI Test Symp. (VTS)*, Apr. 2019, pp. 1–8.

[3] M. Psarakis et al., "Microprocessor software-based self-testing," *IEEE Design Test Comput.*, vol. 27, no. 3, pp. 4–19, May/Jun. 2010.

[4] S. Di Carlo et al., "A software-based self test of CUDA Fermi GPUs," in *Proc. 18th IEEE Eur. TEST Symp. (ETS)*, May 2013, pp. 1–6.

[5] S. D. Carlo, J. E. R. Condia, and M. S. Reorda, "An on-line testing technique for the scheduler memory of a GPGPU," *IEEE Access*, vol. 8, pp. 16893–16912, 2020.

[6] J. E. R. Condia et al., "FlexGripPlus: An improved GPGPU model to support reliability analysis," *Microelectron. Rel.*, vol. 109, Jun. 2020, Art. no. 113660.

[7] P. Bernardi et al., "Development flow for on-line core self-test of automotive microcontrollers," *IEEE Trans. Comput.*, vol. 65, no. 3, pp. 744–754, Mar. 2016.

[8] J.-D. Guerrero-Balaguera, J. E. R. Condia, and M. S. Reorda, "On the functional test of special function units in GPUs," in *Proc. 24th Int. Symp. Design Diag. Electron. Circuits Syst. (DDECS)*, Apr. 2021, pp. 81–86.

[9] J. E. R. Condia and M. S. Reorda, "Testing permanent faults in pipeline registers of GPGPUs: A multi-kernel approach," in *Proc. IEEE 25th Int. Symp. On-Line Test. Robust Syst. Design (IOLTS)*, Jul. 2019, pp. 97–102.

[10] J. E. R. Condia and M. S. Reorda, "Modular functional testing: Targeting the small embedded memories in GPUs," in *Proc. VLSI-SoC (Design Trends Series)*, 2021, ch. 10.

[11] F. A. da Silva et al., "Determined-safe faults identification: A step towards ISO26262 hardware compliant designs," in *Proc. IEEE Eur. Test Symp. (ETS)*, May 2020, pp. 1–6.

[12] Y.-C. Chang et al., "Assessing automotive functional safety microprocessor with ISO 26262 hardware requirements," in *Proc. Tech. Papers Int. Symp. VLSI Design, Autom. Test*, Apr. 2014, pp. 1–4.

Josie E. Rodriguez Condia is interested in functional testing, parallel architectures, and embedded system design. Rodriguez Condia has a PhD in computer engineering from the Politecnico di Torino, 10129 Turin, Italy, and an MSc in electronics from the Universidad Pedagógica y Tecnológica de Colombia (UPTC), Tunja, Colombia. He is a Member of IEEE.

Felipe Augusto da Silva is pursuing a PhD in functional safety with Cadence Design Systems, 85622 Munich, Germany, and the Delft University of Technology, 2028 Delft, The Netherlands. He works on functional safety projects for the automotive, aerospace, and defense industries. Da Silva has an MSc in electrical and electronics engineering from the Federal University of Santa Catarina (UFSC), Florianópolis, Brazil.

Ahmet Çağrı Bağbaba is with Cadence Design Systems, 85622 Munich, Germany. His research interests include hardware functional safety verification in the context of ISO26262, a digital and embedded system design. Bağbaba has a PhD in computer and systems engineering from the Tallinn University of Technology, Tallinn, Estonia, and an MSc in electronics and telecommunication engineering from Istanbul Technical University, İstanbul, Turkey.

Juan-David Guerrero-Balaguera is pursuing a PhD with the Department of Control and Computer Engineering, Politecnico di Torino, 10129 Turin, Italy. His research interests include functional testing, artificial intelligence, and parallel architectures. Guerrero-Balaguera has an MSc in electronics from the Universidad Pedagógica y Tecnológica de Colombia (UPTC), Tunja, Colombia. He is a Member of IEEE.

Said Hamdioui is the Chair Professor and the head of the Department of Quantum and Computer Engineering, Delft University of Technology, 2028 Delft, The Netherlands. His research interests include hardware dependability and emerging computing paradigms. He is a Senior Member of IEEE and serves on the Editorial Board of *IEEE Design&Test*.

Christian Sauer is the head of the European System Design Enablement Team, Cadence Design Systems, 85622 Munich, Germany. He works on customer-specific projects developing tailored solutions for cutting-edge SoCs and systems across automotive and 5G domains. His research interests include the development of application-specific multiprocessor platforms, tools, and methodologies for their applications.

Matteo Sonza Reorda is a full professor with the Department of Control and Computer Engineering, Politecnico di Torino, 10129 Turin, Italy. His research interests include the design and test of reliable electronic circuits and systems. Sonza Reorda has a PhD in computer engineering from the Politecnico di Torino. He is a Fellow of IEEE.

■ Direct questions and comments about this article to Josie E. Rodriguez Condia, Politecnico di Torino, 10129 Turin, Italy; josie.rodriguez@polito.it.

T-Topology Coupler-Based Bandpass Negative Group Delay Active Circuit Design and Test

Taochen Gu and Fayu Wan
School of Electronic and Information Engineering
Nanjing University of Information Science & Technology (NUIST)
Nanjing 210044, China

Jingjie Zhou
Nanjing Institute of Measurement and Testing Technology
Nanjing 210049, China

Qizheng Ji
National Key Laboratory on Electromagnetic Environment Effects
Army Engineering University of PLA
Shijiazhuang 050003, China

Binhong Li
Key Laboratory of Silicon Device Technology
Chinese Academy of Sciences
Beijing 100029, China

Blaise Ravelo
School of Electronic and Information Engineering
Nanjing University of Information Science & Technology (NUIST)
Nanjing 210044, China

Editor's notes:
This article develops a design method for the bandpass negative group delay active microwave circuits. The theoretical calculations, simulation results, and measurements on a tested prototype show excellent consistency.
—Haralampos Stratigopoulos, Sorbonne Université, CNRS, LIP6

THE COUNTERINTUITIVE NEGATIVE group delay (NGD) function was theoretically and experimentally investigated with low-frequency circuit topologies in the 1990s [1]. The meaning of the NGD function was experimentally interpreted with the counterintuitive propagation of output signal raising and trailing edges in advance compared to its input [2]. This extraordinary effect does not contradict the causality principle. Then, the orthodoxly circuit theory with the revelation of basic NGD circuit topologies was introduced [3]. To further understand the NGD function interpretation, a circuit theory emphasizing the similitude with the analog filter behavior was proposed [3]. Because of its counterintuitive effect, the unfamiliar NGD effect causes skepticism in the circuit engineering communities. Since the first experimentation, the NGD topic attracted the curiosity of a few electronic and RF/microwave researchers. One of the remarkable periods was the early 2000s where the NGD function disclosure coincided with the RF and microwave metamaterial revolution [4]. Then, the left-hand-based NGD circuit was validated with different analytical approaches as the consideration of resistive loss [5].

Digital Object Identifier 10.1109/MDAT.2022.3202858
Date of publication: 29 August 2022; date of current version: 10 March 2023.

However, the metamaterial aspects based on NGD passive circuits suffer from excessively high attenuation losses that may reach 20 dB [4] with only a single cell. The circuit loss problem was also found with the resonant resistor–inductor–capacitor (RLC)-network nonmetamaterial NGD circuits. In other words, the NGD values become insignificant if the attenuation loss is excessively high. This technical weakness of NGD functions raises curious questions about the general applications as proposed in [6], [7], and [8]. The tentative fields of application concern group delay (GD) equalization, bilateral gain compensated circuits, frequency-independent phase shifter, and nonfoster element design. To solve the problems caused by the insertion loss, more research work efforts have been made since the early 2010s on the design of RF and microwave NGD active circuits [9], [10], [11]. These existing lumped NGD active circuits are susceptible to compensate for the attenuation loss but their constituting lumped RLC networks cannot work at a high frequency above gigahertz. Thus, the present research work focuses on the design of active and T-topology bandpass (BP) NGD function with the transmission line (TL) theory. The main novelty and contribution of the article in difference from the existing NGD work [11] are

- Theorization of the distributed and active BP NGD topology using a coupled line (CL)-based T-cell passive topology. The overall BP NGD circuit under study is comprised of a distributed passive T-cell compensated by a microwave amplifier.
- Design method of the active circuit in terms of the available expected BP NGD specifications. Choice and integration of low noise amplifier (LNA) to target NGD specifications (NGD center frequency, NGD value, and NGD bandwidth) and loss compensation.

Design, simulation, and experimental validations of the NGD T-topology

This section introduces the S-matrix modeling of the passive T-topology. The BP NGD analysis is performed in function of the circuit parameters.

T-Topological description

Figure 1a shows the equivalent circuit of the T-topology that acts as a two-port circuit built with fully distributed passive structures: Two identical CLs denoted CL1 and CL2 with identical characteristic impedance Z, attenuation a, propagation delay τ, and coupling coefficient k. In Figure 1a, CL1 and CL2 are references with ports, (①, ③, ④, ⑤) and (②, ③, ⑥, ⑦), respectively. An open-ended stub is represented by a TL having identical characteristic impedance Z, attenuation a, and propagation delay τ. As seen in Figure 1a, this TL is connected between port ⑤ and port ⑦. The global S-matrix model is determined based on the equivalent circuit. As introduced in Figure 1b, the TL is reduced as a parallel impedance Z_{in}. According to the TL theory, by denoting the terminal load reference impedance, $R_0 = 50\ \Omega$, $s = j\omega$ the Laplace variable and the angular frequency variable ω, the TL input impedance is given by

$$Z_{\text{in}}(s) = R_0^2 / Z\left[1 - a^2 \exp(-2s\tau)\right] / \left[1 + a^2 \exp(-2s\tau)\right]. \quad (1)$$

Magnitudes of the T-topology reflection and transmission coefficients

Similar to the classical microwave circuit analyses, before the NGD analysis, it is crucial to perceive the frequency responses of the T-topology S-parameter coefficients. Accordingly, the associated magnitudes of the transmission coefficients are

$$S_{21}(\omega) = |S_{21}(j\omega)| = \frac{2R_0(1-k^2)\sqrt{\begin{matrix}[a^2\sin^2(\omega\tau) + \\ a^2\cos^2(\omega\tau) - 1]^2 \\ + a^2\sin^2(\omega\tau)\cos^2(\omega\tau)\end{matrix}}}{\zeta(\omega)} \quad (2)$$

with

$$\zeta(\omega) = \sqrt{\begin{matrix}\left\{\begin{matrix}2R_0(1-k^2) + k^2 Z + a^2\cos(2\omega\tau) \\ \left[k^2(2R_0 + Z) - 2R_0\right]\end{matrix}\right\}^2 \\ + a^4\sin^2[\theta(\omega)]\left[k^2(2R_0 - Z) + 2R_0\right]^2\end{matrix}} \quad (3)$$

where k is the coupling coefficient. We recall that the phase shift associated with the transmission

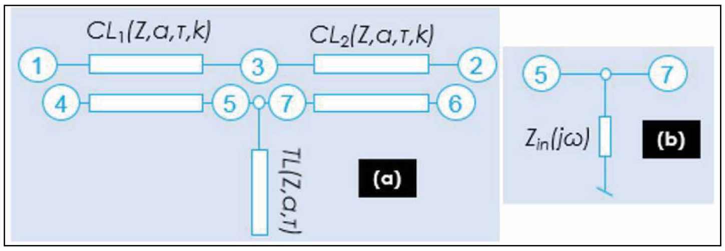


Figure 1. (a) Configuration of the T-topology under study. (b) Open stub input impedance.

coefficient is defined by $\varphi(\omega)=\arg[S_{21}(j\omega)]$. According to the circuit and system theory, the GD can be derived from the transmission coefficient as follows:

$$\mathrm{GD}(\omega)=-\partial\varphi(\omega)/\partial\omega\,. \tag{4}$$

Knowing the transmission phase, the reverse T-topology GD can be analytically calculated from this previous expression.

NGD analysis at very low frequency

One of the natural particular frequencies, which can be investigated for the NGD existence, is the lowest frequency value. The NGD can be analyzed from the GD in (4). At very low frequency, $\mathrm{GD}_0=\mathrm{GD}(\omega\approx 0)$, the reverse T-topology presents the following GD:

$$\mathrm{GD}_0=\frac{-2\pi a^2k^2Z/\omega_0/(1-a^2)}{2R_0(1-a^2-k^2+a^2k^2)+Z(a^2+k^2)^2}\,. \tag{5}$$

We emphasize that this GD can be negative if

$$2R_0(1-a^2-k^2+a^2k^2)+Z(a^2+k^2)^2>0\,. \tag{6}$$

Therefore, the topology under study can behave as a low-pass NGD circuit but this type is out of the range of the present study.

NGD analysis at resonance frequency

The second particular frequency of the topology under study can be resonance $\omega=\omega_0$. At this frequency, the GD is transformed as

$$\mathrm{GD}=\mathrm{GD}(\omega_0)=\frac{2\pi a^2k^2Z/\omega_0/(1+a^2)}{2R_0(k^2-a^2-1+a^2k^2)+k^2Z(a^2-1)}\,. \tag{7}$$

To realize a BP NGD function, the following condition must be satisfied:

$$Z>Z_{\min}=2R_0(k^2-a^2-1+a^2k^2)/\left[k^2(1-a^2)\right]. \tag{8}$$

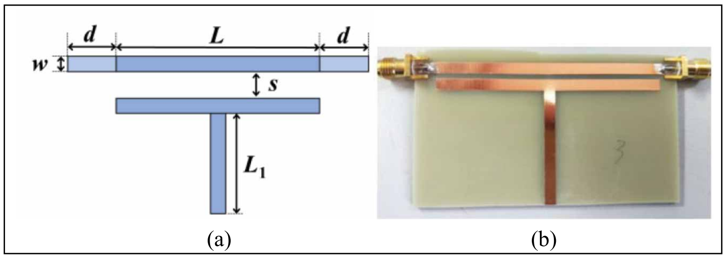


Figure 2. Fabricated NGD passive circuit prototype (a) layout and (b) photograph.

In addition, the reflection and transmission coefficients are expressed as

$$S_{11}^{T}(\omega_0)=\frac{k^2Z(1-a^2)}{\left|2R_0(k^2-a^2-1+a^2k^2)+k^2Z(a^2-1)\right|} \tag{9}$$

$$S_{21}^{T}(\omega_0)=\frac{2R_0(1+a^2)(1-k^2)}{\left|2R_0(k^2-a^2-1+a^2k^2)+k^2Z(a^2-1)\right|}\,. \tag{10}$$

These S-parameters are susceptible to satisfy the constraints $S_{11}^{T}(\omega_0)<10^{-10\mathrm{dB}/20}$ and $S_{21}^{T}(\omega_0)\neq 0$.

To more realistically verify the efficiency of the developed BP NGD theory, the proof-of-concept (POC) will be investigated in the following section.

Design description of BP NGD T-circuit prototype

By using the previous relations, the theoretical model of the T-structure can be compared with simulation and measurement. Doing this, as POCs, passive and active BP NGD circuits were designed, simulated, fabricated, and tested to verify the validity of the theory established in the following section. The design process of the BP NGD CL-based T-circuit was similar to the classical electronic circuits (filters, phase shifters, couplers, power dividers, and so on). All simulation results in this article are obtained from simulations with the microwave electronic circuit designer and simulator advanced design system (ADS) from Keysight Technologies. The measurements were performed with a vector network analyzer (VNA). Figure 2a and b introduces the ADS design layout and photograph of the T-microstrip circuit.

This passive circuit was implemented in fully distributed microstrip technology without using lossy lumped circuits. The fabricated prototype that is displayed in Figure 2 has a physical size of 40 mm × 80 mm. The prototype was realized on the FR4 substrate with the characteristics in Table 1. Before the fabrication, the TL and CL were slightly optimized to reach better NGD performances. It should be emphasized that the effect of TL connecting the access ports is negligible because of the well-matching effect, and its GD is notably small compared to the targeted NGD value as indicated in Table 1. The TL and CL physical widths w correspond to characteristic impedance $Z=48\ \Omega$. The TL and CL quarter wavelength ($\theta=90°$) is set at the NGD center frequency $f_0=1.35$ GHz. The considered CLs have identical coupling coefficients $k=-15$ dB.

Passive T-prototype modeled, simulated, and experimental results

To experimentally validate the NGD function with the investigated T-topology, the prototype in Figure 2 was tested. Then, the results from the MATLAB program, ADS simulations, and measurements will be compared and discussed. The NGD prototype was measured using a VNA provided by Rohde & Schwarz (ZNB 20, frequency band from 100 kHz to 20 GHz). The comparative results are obtained from 1.2 to 1.4 GHz as depicted in Figure 3. As expected, this result proves the validity of the BP NGD function generated by the T-topology introduced in Figure 1. The NGD prototype has a bandwidth of approximately 24 MHz. As plotted in Figure 3a, the NGD optimal value is approximately –14.3 ns in simulation versus –13 ns in measurement. The NGD center frequency is approximately 1.36 GHz. Table 2 summarizes the comparison of NGD performances from the model, simulation, and measurement. Slight differences in GD calculated from (7), simulations, and experimental results, notably at approximately the NGD center frequency, are observed. They are notably due to the dielectric substrate permittivity tolerance, the substrate dispersion loss, the metallization skin effect, and the fabricated circuit imperfection in the considered working frequency. The relative permittivity and loss tangent of PCB may vary with frequency, temperature, and other factors. Therefore, there is a slight difference between the simulated and measured frequencies. Moreover, Figure 3b introduces that the designed NGD prototype ensures –9 dB transmission coefficients in simulation and measurement at approximately the center frequency.

Additionally, as depicted in Figure 3c, the reflection coefficient is better than –17.7 dB within the NGD bandwidth.

Table 1.NGD T-circuit parameters and specifications.

Components	*Description*	*Parameter*	*Value*
Dielectric substrate	Material	FR4	-
	Relative permittivity	ε_r	4.8 @1MHz
			4.6 @1GHz
	Loss tangent	tan(δ)	0.015 @1MHz
			0.020 @1GHz
	Thickness	h	0.8 mm
Metallization	Material	Copper (Cu)	-
	Thickness	t	35 µm
	Conductivity	σ	58 MS/m
TL$_1$	Total length	L_1	30 mm
	width	w	1.5 mm
	attenuation loss	a_1	-0.08 dB
	delay	τ_1	0.185 ns
CL	length	L	30 mm
	width	w	1.5 mm
	Interspace	s	0.5 mm
	Coupling coefficient	k	-14.7 dB
Access line	length	d	10 mm
	width	w	1.5 mm

BP NGD experimental analysis of active topology

To compensate for the insertion loss of the passive NGD circuit introduced in Figure 1, we have fabricated another active circuit prototype using a microwave amplifier which is shown in Figure 4. In this case, the POCs are constituted by a passive T-circuit and a packaged LNA. The employed amplifier is a surface-mounted monolithic LNA referenced LEE-9+ from mini-circuits whose gain is 8.5 dB. The experimental validation of the BP NGD active circuit was also performed based on S-parameter measurements. The VNA from Rohde & Schwarz is defined by the specification ZNB 20 frequency band from 100 kHz to 20 GHz. During the test, the active circuit was biased with a $V_0 = 5\ V_{DC}$ power supply.

Discussion on NGD results of the active circuit test

The simulation and measurement results are compared in Figure 5. The simulation result was performed from S-parameters in the ADS environment. During the simulation, the touchstone file of LEE-9+ provided by the manufacturer [12] was considered. The measured frequency responses are very well correlated with the simulations. Furthermore, the results confirm the BP NGD function without loss expected with the T-topology and a microwave amplifier. The amplifier does not affect the GD responses.

As shown in Figure 5a and b, NGD performances from measurement and simulations present an excellent consistency in terms of simulated NGD center frequency, value, and bandwidth. The relative inaccuracies of the reflection coefficient between measurements and simulations are 5% better than the expected values.

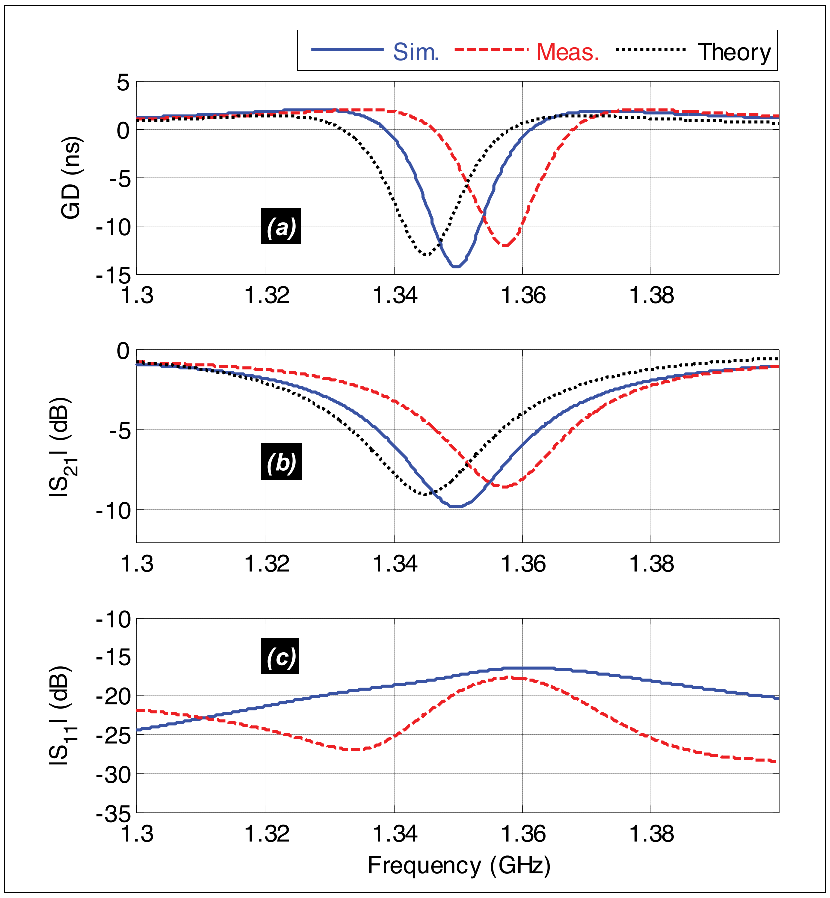


Figure 3. (a) GD, (b) transmission, and (c) reflection coefficients of the fabricated T-circuit, as shown in Figure 2.

Table 2. Calculated, simulated, and experimented NGD performances.

Validation Method	f_0 (GHz)	GD_n (ns)	*BW* (MHz)	S_{21} (dB)	S_{11} (dB)
Theory	1.345	-12.1	26	-8.5	-35.1
Simulation	1.350	-14.3	22	-9.8	-16.5
Measurement	1.357	-13	24	-9.0	-17.7

Active NGD circuit specifications

In addition to the NGD analyses, the power added efficiency (PAE) and intercept products were also characterized. As discussed in the following paragraphs, the developed NGD circuit presents an interesting active performance.

a) PAE characterization of the active NGD circuit

The following parameters are considered to perform the NGD circuit PAE analysis. We denote

- the power consumption, P_{DC}(NGD);
- the root mean square (RMS) input power, P_{in}(NGD); and
- the RMS output power, P_{out}(NGD).

The characterization was based on the NGD circuit output with a 1 dB compression point denoted $P_{out\text{-}1\ dB}$. The NGD circuit PAE was assessed from the relation

$$\mathrm{PAE(NGD)} = \frac{P_{out}(\mathrm{NGD}) - P_{in}(\mathrm{NGD})}{P_{DC}(\mathrm{NGD})}. \qquad (11)$$

During the test, the bias voltages are set to V_{GG} = 9 V. Figure 6 shows the results of the $P_{1\ dB}$ and PAE characterization of the active NGD circuit. Figure 6a shows the output 1 dB compression point, and $P_{out\text{-}1\ dB}$ = 15.4 dBm is measured at 1.36 GHz. The variation is widely less than 0.5 dB from 1.2 to 1.5 GHz. This 1-dB compression depends mainly on the used LNA. When the power consumption of the active NGD circuit is equal to 450 mW, as shown in Figure 6b, the NGD circuit PAE is approximately 7.5%. This PAE is almost insensitive to the frequency from 1.2 to 1.5 GHz. This NGD circuit PAE can be improved by using less power-consuming NGD passive topology.

b) Input/output intercept products characterization of the active NGD circuit

The nonlinearity characterization of the NGD circuit is based on the assessment of the output and input third-order intercept products (OIP3 and IIP3). Two different sine signals

$$\begin{cases} v_1(t) = V_{1\max} \sin(2\pi f_1 t) \\ v_2(t) = V_{2\max} \sin(2\pi f_2 t) \end{cases} \tag{12}$$

with frequencies f_1 and f_2 were injected to realize this nonlinearity analysis. The two-input signal amplitudes were maintained

$$V_{1\max} = V_{2\max} = V_{\max}. \tag{13}$$

We denoted the amplitude of input signal fundamentals as P_1, and the output one P_2 corresponds to the frequency

$$f_{out} = 2f_1 - f_2. \tag{14}$$

The output third-order intercept was calculated by

$$\text{OIP3(NGD)} = \frac{P_1 - P_2}{2} + P_1. \tag{15}$$

By denoting the NGD circuit gain by G, we extract the input third-order intercept from the equation

$$\text{IIP3(NGD)} = \text{OIP3(NGD)} - G. \tag{16}$$

The measurement test setup is shown in Figure 7a. The following equipment was used during the test.

- Two signal generators with reference: 1) Agilent MXG Analog Signal Generator N5183A operating in the bandwidth of 100 kHz–40 GHz and 2) Agilent N9310A RF Signal Generator with the frequency band of 9 kHz–3 GHz to synthesize two-tone harmonic signals with center frequencies: $f_1 = f_0$ and $f_2 = f_0 + 1$ MHz with $f_0 = 1.35$ GHz.
- The two-tone signals were combined using a power combiner, referenced, ZFRSC-42-S+, which operates from DC-4200 MHz provided by a mini-circuit.
- The signal spectra are visualized with the spectrum analyzer reference, Agilent MXE EMI Receiver N9038A, which operates from 20 Hz–26.5 GHz.

The measured OIP3 and IIP3 are displayed in Figure 7b. We can emphasize that OIP3 is 29 dBm, and IIP3 is 21 dBm.

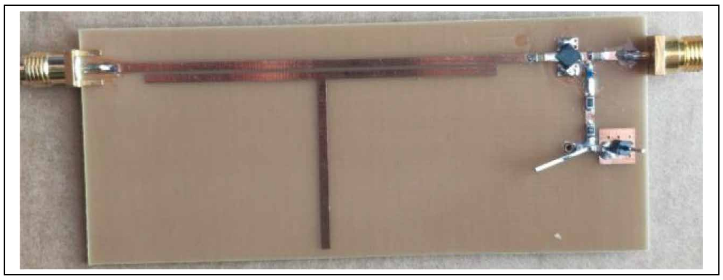


Figure 4. Photograph of the fabricated active NGD circuit.

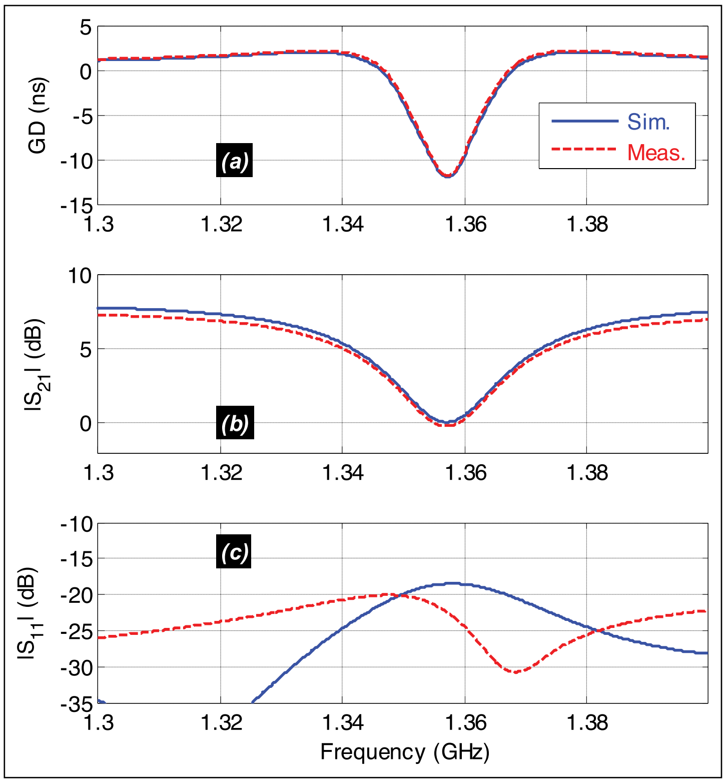


Figure 5. (a) GD, (b) transmission, and (c) reflection coefficients of the fabricated NGD circuit in Figure 4.

Discussion on the performance of an active prototype NGD compared with the literature

Table 3 summarizes the comparison of NGD performance parameters: f_0, GD_n, BW, and $S_{21}(f_0)$. Table 3 shows the performance results of the proposed NGD active circuit compared to others in the literature [9], [10], [11], [13], [14], [15]. Subsequently, the introduced T-structure and LNA-based NGD topology have the main advantages in terms of:

- possibility to operate with a large NGD value;
- low signal attenuation;

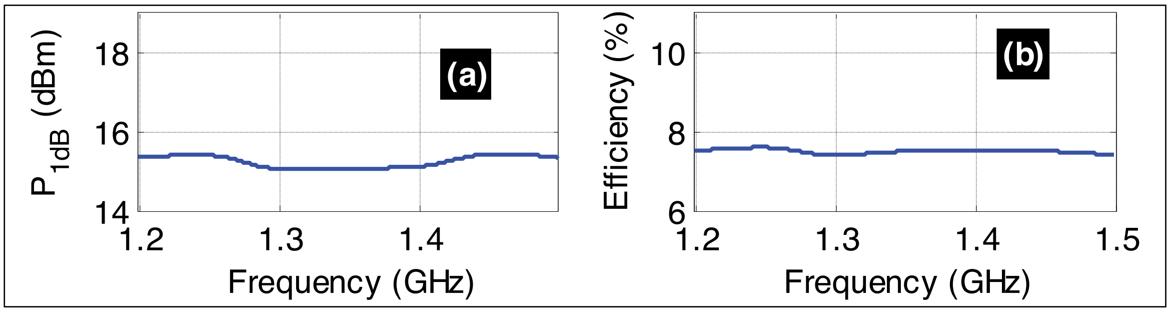


Figure 6. Experimental (a) 1 dB power compression point and (b) PAE for the active NGD circuit.

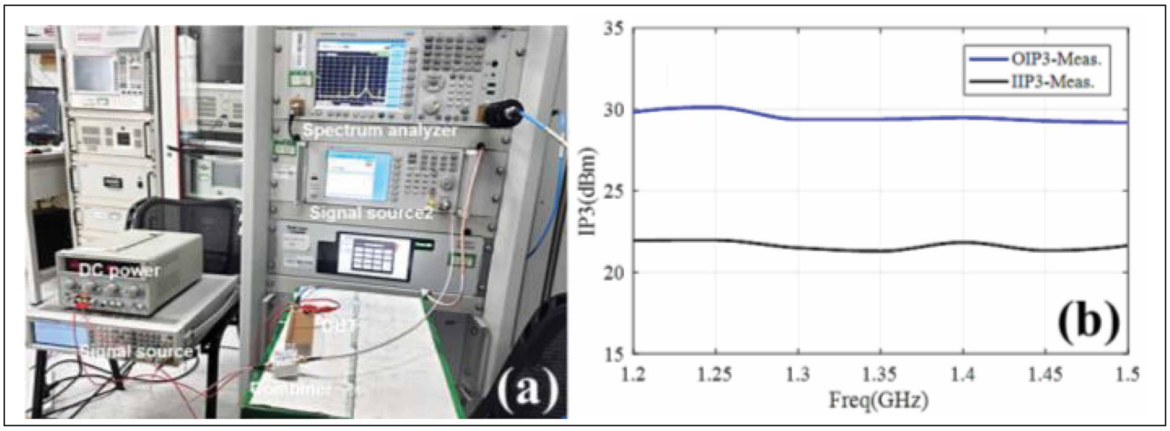


Figure 7. OIP3 and IIP3 characterization. (a) Experimental setup. (b) Measured results.

Table 3. NGD performance comparison.

References	f_0 (GHz)	GD_n (ns)	BW (MHz)	$S_{21}(f_0)$ (dB)	$S_{11}(f_0)$ (dB)	Implementation technology	Nonlinear (NL) and noise figure (NF) characterization?
[6]	0.31	-1.86	20	-0.29	-18	SMC R, L and C component-based circuit	No
[7]	0.915	-0.03	500	-12.2	-25	SMC R, L and C component-based circuit	No (only NF of the application amplifier provided)
[9]	1	-1	200	0	-25	Distributed and active	No
[10]	1.15	-3	75.9	0	-30	SMC R, L and C component-based circuit	No
[11]	0	-5	32	0	-8	SMC R, L and C component-based circuit	No (RLC-network lumped active circuit without NL and NF characterization)
[13]	1.00	-2.06	40	-7.8	-25.7	Distributed and passive circuit	No
[14]	1.016	-2.09	144	-18.1	-33	Distributed and passive circuit	No
[15]	2.303	-8.1	27	-6.66	-5.78	Distributed and passive circuit	No
Proposed one	1.357	-13	24	-9.0	-17.7	Distributed and passive circuit	No
	1.36	-13	24	0	-20	Distributed and active circuit	Yes (NGD added with PAE, P1dB, NF and IP3 characterization)

- design simplicity; and
- good return loss.

The main novelty of the present research work, in difference from the NGD design proposed in [11], concerns the nonlinear (NL) and noise figure (NF) characterization of a microwave active circuit implemented with a fully distributed microstrip passive NGD circuit part. Compared with other passive NGD circuits [13], [14], [15], the passive structure proposed in this work has the advantage of larger NGD value, good return loss, and design simplicity. Correspondingly, the bandwidth of T-topology is relatively smaller than that of other passive structures.

The proposed circuit is promising to be useful but for many wireless communication channels as in standard IEEE 802.11b, satellite communication channels are limited to 20 MHz, and the proposed circuit bandwidth is good enough.

A BP NGD theory of active microwave circuits including a passive T-topology is developed. The passive T-topology is composed of fully passive distributed elements with a TL and two identical CLs. The S-matrix models of both passive and active topologies are established. More importantly, the calculated, simulated, and measured results were compared. As POC, the tests and validations were performed with an NGD circuit prototype designed and implemented in microstrip technology. Excellent consistency was observed between simulations and measurements, which confirm the BP NGD behavior. The tested prototype has excellent performance compared with those in the literature. The measured GD value of –13 ns and a transmission coefficient better than 0 dB were achieved at the center frequency of approximately 1.36 GHz. ■

Acknowledgments

This work was supported in part by NSFC under Grant 61971230, in part by the Jiangsu Distinguished Professor Program and Six Major Talents Summit of Jiangsu Province under Grant 2019-DZXX-022, and in part by the Startup Foundation for Introducing Talent of NUIST.

References

[1] M. W. Mitchell and R. Y. Chiao, “Causality and negative group delays in a simple bandpass amplifier,” *Amer. J. Phys.*, vol. 66, no. 14, pp. 14–19, 1998.

[2] M. Kitano, T. Nakanishi, and K. Sugiyama, “Negative group-delay and superluminal propagation: An electronic circuit approach,” *IEEE J. Sel. Top. Quantum Electron.*, vol. 9, no. 1, pp. 43–51, Feb. 2003.

[3] B. Ravelo, “Similitude between the NGD function and filter gain behaviours,” *Int. J. Circuit Theory Appl.*, vol. 42, no. 10, pp. 1016–1032, Oct. 2014.

[4] O. F. Siddiqui, M. Mojahedi, and G. V. Eleftheriades, “Periodically loaded transmission line with effective negative refractive index and negative group velocity,” *IEEE Trans. Antennas Propag.*, vol. 51, no. 10, pp. 2619–2625, Oct. 2003.

[5] J. J. Barroso et al., “Negative group velocity in resistive lossy left-handed transmission lines,” *IET Microw., Antennas Propag.*, vol. 11, no. 15, pp. 2235–2240, Oct. 2017.

[6] M. Kandic and G. E. Bridges, “Bilateral gain-compensated negative group delay circuit,” *IEEE Microw. Compon. Lett.*, vol. 21, no. 6, pp. 308–310, Jun. 2011.

[7] Y. Meng et al., “A broadband switch-less bi-directional amplifier with negative-group-delay matching circuits,” *Electronics*, vol. 7, no. 9, pp. 1–11, Aug. 2018.

[8] T. Zhang, R. Xu, and C.-T. M. Wu, “Unconditionally stable non-foster element using active transversal-filter-based negative group delay circuit,” *IEEE Microw. Wireless Compon. Lett.*, vol. 27, no. 10, pp. 921–923, Oct. 2017.

[9] C.-T. M. Wu and T. Itoh, “Maximally flat negative group-delay circuit: A microwave transversal filter approach,” *IEEE Trans. Microw. Theory Techn.*, vol. 62, no. 6, pp. 1330–1342, Jun. 2014.

[10] M. Kandic and G. E. Bridges, “Asymptotic limits of negative group delay in active resonator-based distributed circuits,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 58, no. 8, pp. 1727–1735, Aug. 2011.

[11] F. Wan et al., “The design method of the active negative group delay circuits based on a microwave amplifier and an RL-series network,” *IEEE Access*, vol. 6, pp. 33849–33858, 2018.

[12] *Mini-Circuits Monolithic Amplifier LEE-9+.* Accessed: May 10, 2020. [Online]. Available: https://www.minicircuits.com/WebStore/dashboard.html?model=LEE-9%2B

[13] Z. Zhu et al., “A novel balanced-to-unbalanced negative group delay power divider with good common-mode suppression,” *Int. J. RF Microw. Comput.-Aided Eng.*, vol. 32, no. 7, 2022, Art. no. e23173.

[14] Z. Wang et al., “A negative group delay microwave circuit based on signal interference techniques,” *IEEE Microw. Wireless Compon. Lett.*, vol. 28, no. 4, pp. 290–292, Apr. 2018.

[15] R. Vauché et al., “Experimental time-domain study for bandpass negative group delay analysis with lill-shape microstrip circuit,” *IEEE Access*, vol. 9, pp. 24155–24167, 2021.

Taochen Gu is pursuing a PhD with the Nanjing University of Information Science and Technology (NUIST), Nanjing 210044, China. His research interests include abnormal wave propagation in dispersive media and microwave circuits. Gu received a BSc in electrical engineering from NUIST. He is a Student Member of IEEE.

Fayu Wan is a full professor at the Nanjing University of Information Science and Technology (NUIST), Nanjing 210044, China. His current research interests include negative group delay (NGD) circuits, electrostatic discharge, electromagnetic compatibility, and advanced RF measurement. Wan received a PhD in electronic engineering from the University of Rouen, Rouen, France. He is a Senior Member of IEEE.

Jingjie Zhou is an engineer with the Nanjing Institute of Measurement and Testing Technology, Nanjing 210049, China.

Qizheng Ji is a professor of engineering at the Army Engineering University of PLA, Shijiazhuang 050003, China.

Binhong Li is an associate professor at the Key Laboratory of Silicon Device Technology, Chinese Academy of Sciences, Beijing 100029, China.

Blaise Ravelo is a university full professor at the Nanjing University of Information Science and Technology (NUIST), Nanjing 210044, China. His research interests include multiphysics and electronics engineering. He is a member of IEEE.

■ Direct questions and comments about this article to Fayu Wan, School of Electronic and Information Engineering, Nanjing University of Information Science & Technology (NUIST), Nanjing 210044, China; fayu.wan@nuist.edu.cn.

FPGA-Chain: Enabling Holistic Protection of FPGA Supply Chain With Blockchain Technology

Tao Zhang, Fahim Rahman, Mark Tehranipoor, and Farimah Farahmandi
Department of Electrical and Computer Engineering
University of Florida
Gainesville, FL 32611 USA

Editor's notes:
Field-programmable gate array (FPGA) bitstream reverse engineering and counterfeiting is a pertinent challenge in the modern hardware supply chain. To this end, this article proposes a blockchain-based technology to foster authenticity and integrity of the FPGA supply chain for trustworthy traceability. The proposed approach is transformative in being able to detect counterfeit FPGA chips and bitstreams using state-of-the-art blockchain technologies.

—Kanad Basu, The University of Texas at Dallas

■ **The global FPGA** market size is valued at about $9 billion in 2020 and is projected to grow to $18.8 billion in 2027 due to the incoming Internet-of-Things (IoT) era [1]. Thanks to their excellent flexibility and performance, FPGAs are becoming one of the major IoT platforms to meet the evolving functional, reliability, and security requirements throughout many years of infield execution, supporting the mission-critical infrastructure in hospitals, airports, and military bases [2].

Despite the bright market prospect, the FPGA supply chain is gravely threatened from the trust perspective, that is, counterfeit FPGA hardware devices and software bitstream tampering. A counterfeit FPGA device can be a recycled chip reclaimed from the discarded boards, cloned copies, an item remarked to a higher specification or overproduced product from a rogue foundry [3]. The scenarios get even worsen due to the chip shortage induced by the COVID-19 pandemic which encourages the counterfeiters to flood the supply chain with illegitimate FPGA ICs for higher revenue. The inundation of counterfeit FPGAs would undermine the reliability and trustworthiness of mission-critical infrastructures, leading to disastrous results potentially. Moreover, the malleable configuration bitstream (FPGA firmware encoding the user-defined circuitry) is susceptible to tampering by inserting malicious functionality such as information leakage and denial-of-service [4]. Given the serious security concerns from the counterfeit devices and tampered bitstream threats, a transparent, fair, and the trusted FPGA supply chain is in high demand to ensure the authenticity of purchased FPGA chips and associated firmware.

Although numerous techniques have been proposed against counterfeit devices and bitstream tampering in the past decade, a silver bullet solution is

Digital Object Identifier 10.1109/MDAT.2022.3213998
Date of publication: 12 October 2022; date of current version: 10 March 2023.

still unavailable, unfortunately [5]. For example, to identify recycled and remarked FPGAs, people establish complicated standards and electrical tests which are costly and less effective against counterfeit ICs without observable defects or degradation [3]. Hardware devices and FPGA bitstreams can be protected by logic locking [6] by adding extra key gates in the original netlist to be only unlocked with a correct key. However, SAT attacks [6] as well as advanced optical physical attacks [7] fundamentally challenge the scheme. The conventional original component manufacturer (OCM) query-based system verifies the devices by checking the chip ID from users [8]. If the corresponding static record can be found in the database, the item is recognized as an authentic device. However, the scheme does not provide any traceability or inform the ownership status of the target chip so that an intelligent adversary can easily inject counterfeit devices with legitimate IDs (e.g., recycled devices) to deceive the query mechanism. Considering these limitations, it is imperative to establish a framework to provide holistic protection by fulfilling the following requirements.

- *Traceability empowerment.* Lacking traceability presents formidable challenges to protecting the FPGA supply chain. Empowering traceability could tackle the root causes of illegitimate devices and bitstreams instead of endlessly proposing various offline techniques with limited scopes.
- *Fair data management.* The authentic information of FPGA devices and bitstreams should be stored in a secure database to serve as the foundation of traceability. The information should be managed fairly; single entities cannot stealthily manipulate or delete the records from the database.
- *Effective authentication protocols.* To enable trustworthy traceability in the convoluted supply chain, authentication protocols must be available to ensure 1) every *hardware* FPGA instance can be uniquely identified and authenticated and 2) the *software* bitstream integrity can be verified.

These requirements inspire us to propose our novel blockchain-based framework, FPGA chain, protecting the FPGA supply chain against the intrusions of problematic devices and bitstreams. Blockchain, as a decentralized ledger system, can increase transparency and reduce cost and risks across the supply chain [5], thus becoming our infrastructure choice for FPGA supply chain management. We summarize our contributions below.

- We propose the FPGA-chain framework that can enroll the information of FPGA devices and bitstreams and trace and track devices throughout the diverse supply chain, providing the provenance at any arbitrary stage according to the ownership history, helping identify counterfeit devices.
- We store the data records in the synchronized ledger distributed on multiple peer nodes and manage them with customized smart contracts. Any operations on the ledger need a consensus across organizations, enabling fair data management.
- We propose an authentication protocol covering both device authenticity and bitstream integrity verification utilizing the partial run-time reconfiguration capability available in modern FPGAs [9] to prevent illegitimate components from entering the user domain.

Background

FPGA supply chain and threat model

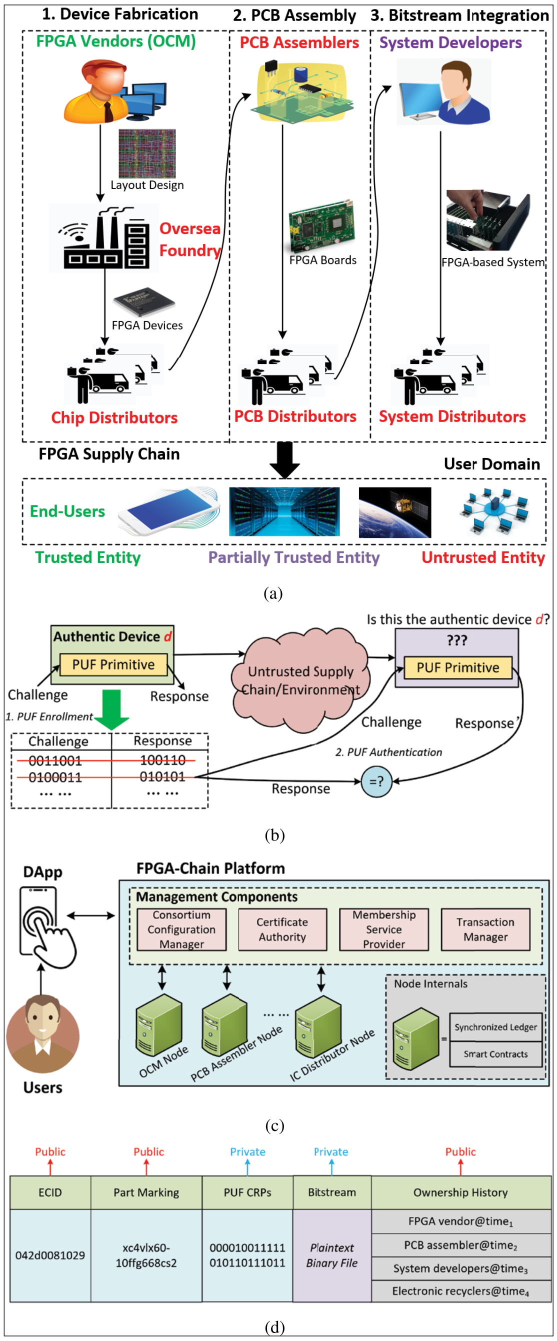
Figure 1a presents the FPGA supply chain and our threat model. A typical FPGA supply chain path starts with the presilicon design from the OCMs, that is, FPGA vendors like AMD-Xilinx and Intel. Their engineers hand the layout design to the (offshore) foundries for device fabrication and packaging. The product chips then enter the chip distribution network from where PCB assemblers can purchase these items. The FPGA boards mounted with peripheral modules such as power management units and external memories can be later distributed by PCB distributors. System developers purchase the board and program the target FPGA by installing the configuration bitstream. Such FPGA-based systems are distributed by system distributors and finally reach the end-users for various applications. As FPGAs would be integrated into mission-critical systems such as military infrastructure or medical equipment, end-users always want authentic devices and compliant bitstreams to protect the final systems from reliability and/or security issues. OCMs also need a genuine supply chain to protect their revenue and reputations while the system developers expect their asset bitstreams should not be tampered with and conform to the original specification. Therefore, in our threat model (see also Figure 1a), except for OCMs and end-users, other participants can be untrusted. The system developers are *partially trusted* since they might introduce counterfeit devices but have no motivation to compromise their own bitstreams. We articulate the targeted threats below.

- *Recycled devices:* Untrusted IC/PCB/system distributors, PCB assemblers, and system developers might intentionally introduce recycled FPGAs (e.g., reclaimed from discarded boards) as new ones for cost savings or higher profits.
- *Overproduced devices:* Rogue foundries have the access to the layout design and can fabricate more items than ordered. These illegitimate copies can be sold to untrusted entities in the supply chain through the gray market.
- *Remarked devices:* The marking information on the chip surface indicates the part name, speed grade, lot number, and so on. An untrusted entity can remark an FPGA to a higher grade to extend the profit margin.
- *Cloned devices:* A cloned device is a replica of the authentic device through hardware reverse engineering and refabrication. The untrusted entities could inject these illegitimate devices stealthily, whereas cloned items potentially have serious quality and reliability issues.
- *Tampered bitstreams:* The configuration bitstreams determine the FPGA in-field behaviors. They can be manipulated by the untrusted system distributors for inserting malicious functionality, compromising the confidentiality, integrity, and availability of the final systems.

Given the above-mentioned threats, we propose our blockchain-centric framework, FPGA chain, to address them by empowering device traceability and authenticity verification in the supply chain.

Existing supply chain protection schemes

Counterfeit detection techniques involve sophisticated (even destructive) inspections such as electronic tests, optical, and T-Hz imaging, which are typically expensive and expertise-intensive [3]. Counterfeit avoidance primitives are a set of low-cost primitives that can help measure chip status and/or fingerprint silicon [3]. Among them, physical



ECID	Part Marking	PUF CRPs	Bitstream	Ownership History
042d0081029	xc4vlx60-10ffg668cs2	000010011111 010110111011	*Plaintext Binary File*	FPGA vendor@$time_1$
				PCB assembler@$time_2$
				System developers@$time_3$
				Electronic recyclers@$time_4$

Figure 1. (a) FPGA supply chain and our threat model, (b) PUF application for counterfeit device detection, (c) consortium blockchain-based FPGA-chain framework, and (d) example entry for an FPGA object in the distributed blockchain ledger.

unclonable function (PUF) primitives can translate the silicon process variations to chip-unique signature, that is, challenge-response pairs (CRPs), as shown in Figure 1b. The CRPs are profiled and stored by OCMs in a secure database during the PUF enrolment phase and can be invoked for detecting counterfeit devices by checking the difference from the response of the same challenge at an arbitrary stage of the supply chain. Note that the used CRPs will expire and need to be deleted from the database. PUF primitives are, albeit effective, not available in most commercial FPGAs. As for FPGA bitstream protection, encryption and authentication are the most prevalent scheme where the adversary is supposed to access the ciphertext data only. However, some recent research has demonstrated how to exploit the vulnerabilities in several FPGA families to decrypt, manipulate, and re-encrypt the ciphertext bitstreams without knowing the symmetric key [9]. Some OCMs offer the static ID-based verification mechanism which searches for the user-provided ID in their centralized database for authenticity validation. However, a counterfeit device such as a recycled or cloned chip can have a legitimate ID and bypass this test. Besides, this mechanism suffers from single-point failure and insider data manipulation risks where an administrator can arbitrarily add/change/delete data records stealthily. In contrast, blockchain emerged as a decentralized and transparent infrastructure, becoming a perfect candidate for enabling trustworthy traceability throughout the supply chain.

Blockchain for electronic supply chain

Blockchain emerges as the promising platform for the supply chain scheduling by storing and managing the data in a decentralized ledger. Unlike the traditional centralized database which is managed by a central administrative authority, blockchain relies on a certificate authority (CA) network which is a group of verified peer nodes that jointly approve new operations based on the underlying voting consensus mechanism. The peer-to-peer topology of blockchain allows the ledger to be stored synchronously on millions of servers. Blockchain infrastructure can be classified into three categories: 1) public; 2) private; and 3) consortium. Public blockchain allows anyone to view and participate in the infrastructure (e.g., Bitcoin) while private blockchain is confined to those nodes which are permissioned by the administrative entity, which works well inside a single organization with multiple departments. Considering the FPGA supply chain involves numerous different companies/entities, the consortium style is a good fit since it is a permissioned platform governed by multiple organizations, combining the merits of both private (efficiency) and public (decentralization) variants.

Although blockchain has been used for scheduling the general microelectronic supply chain in [5], [8], and [10], the FPGA supply chain is confronted with additional vulnerabilities from the software bitstream perspective. Besides, [5] and [10] assume there are available hardware PUF instances on the target silicon for authentication, which is not the case for most commercial FPGAs. Therefore, our FPGA-chain framework is tailored to suit the fact much better by using soft PUFs and run-time configuration data verification which will be detailed in the upcoming section.

FPGA-chain framework

Overview

The overview of our proposed FPGA-chain framework is depicted in Figure 1c. There are management components of the platform.

- *Consortium configuration manager:* It defines the policies of the FPGA-chain platform. For example, an asset object can only be created by OCMs and updated upon consensus.
- *CA:* It cryptographically ensures the confidentiality and integrity of each transaction and operation for peer nodes.
- *Membership service provider:* It assigns and manages the identities of involved members.
- *Transaction manager:* It interacts with the smart contracts for receiving requests, schedules the consensus voting among peer nodes, and post-transactions to all synchronized ledgers. Note that, instead of using the computationally intensive proof-of-work (PoW) protocol, we update the ledger through the peer-level verification of transaction correctness that can be implemented using the practical byzantine fault-tolerate (PBFT) algorithm and digital signatures for better efficiency and throughput.

These management components control and manage the peer nodes which are deployed on the hosts of OCMs, PCB assemblers, system developers, and IC/PCB/system distributors. Each node will have a copy of the synchronized ledger containing identical records. Also, there are smart contracts that are service programs to be invoked by users through distributed applications (DApps) on their terminals. The available smart contracts are the following.

- *Check_ID:* Only OCMs can invoke the contract to verify whether the incoming chip ID is presented in the processing wish list.
- *Create_Asset:* Only OCMs can invoke the contract to create a new data object for the target FPGA (asset) in the distributed ledger (see Figure 1d for an example entry).
- *Update_Asset*: This contract is executed to update the information such as properties or ownership status of an asset based on the consensus of peer nodes.
- *Verify_Asset:* This contract can be invoked by anyone in the supply chain to trace and track the provenance of the target FPGA to avoid counterfeit scenarios.
- *Authenticate_Asset:* Only end-users can invoke the contract to receive a partial bitstream through DApp that can be loaded on their FPGAs and return the outcome packet to the FPGA chain for authenticating the hardware device and/or software bitstream.

Device enrolment

After device fabrication, the FPGA ICs should be sent to a trusted (in-house) facility for device enrolment to avoid potential errors and security risks (see Figure 2a). The facility will send the IC enrolment request through DApp that is later approved by the OCM. Next, a unique electronic chip ID (ECID) should be sent to the OCM as the identifier. ECID, as a common scheme for IC identification, is programed into each die during the wafer test. For example, every Xilinx FPGA is assigned with a 57-bit read-only **Device_DNA** that is accessible through either the JTAG interface or the internal **DNA_Port** primitive.[1] OCM will execute the contract *Check_ID* to see whether this ECID belongs to one of the chips to be enrolled. If yes, an FPGA bitstream encoding a PUF implementation will be transferred to the facility that can be downloaded to the target device for collecting CRPs. As detailed in the previous section, PUF is used to fingerprint every silicon by utilizing the process variations. However, given the unavailability of hardware PUFs in most modern FPGAs, we propose that OCMs could develop an FPGA *soft* PUF at RTL or gate-level [11] to implement the primitive on the configurable fabric with good metrics like uniqueness, stability, uniformity, and area overhead, as an alternative. The facility can use the soft PUF to generate the CRPs and send them back along with the part marking and ECID to OCMs. With the information, OCMs can execute the *Create_Asset* to initiate the ledger entry for a new FPGA object as illustrated in Figure 1d. The blue fields will be filled with ECID, part marking, and PUF CRPs, while the ownership history includes the current owner (OCM) and the exact timestamp. Note that the PUF CRPs serve as the root-of-trust for the device authentication, so the data privacy policy will keep them private, that is, only visible to the trusted OCMs, whereas ECID and part marking can be public for identification. These privacy policies enforce necessary access control to constitute fair data management in the FPGA chain, effectively avoiding sensitive information leakage to untrusted entities.

Transactions between supply chain entities

Despite the convoluted FPGA supply chain, most of the transactions can be considered in a similar model from the FPGA-chain side, that is, the buyers join the FPGA chain, get the IDs from the member service provider, and place orders through the DApp. The transaction manager component will schedule these orders and distribute them to the corresponding sellers. After receiving the items, the buyer will verify the devices based on the ECIDs. Successful verification will motivate the buyers to confirm the items and the ownership status of the item needs to be updated. This flow is fully compatible with the present trading routine, while the sensitive business statistics such as batch number and price can be visible to permissioned entities under proper access control policies. Here, we highlight a typical case that a system developer wants to purchase an FPGA board from a PCB assembler as shown in Figure 2b. The FPGA device on the board is acquired from an IC distributor and enrolled by the OCM. The system developer sends the board purchase request via DApp and receives the boards shipped from the PCB assembler. The ECIDs can be read from the received FPGAs as the input to the smart contract *Verify_Asset* that will check whether the current owner of the ECID is consistent with the identity of the PCB assembler,

[1]We will mostly use Xilinx terminology (such as **Device_DNA**) for simplicity but the components like embedded ECID and partial reconfiguration capabilities are universal across mainstream vendors.

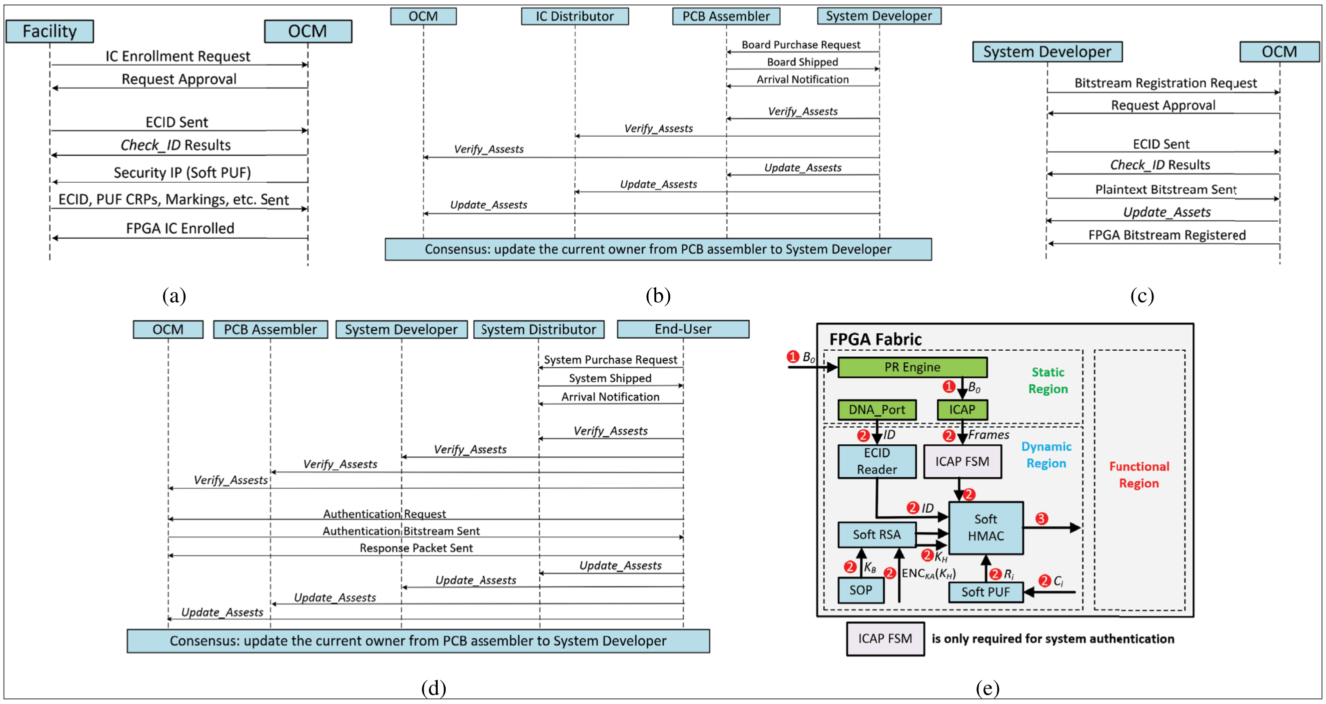


Figure 2. (a) Device enrolment. (b) Transactions between supply chain entities. (c) Bitstream registration. (d) Entering the user domain. (e) Authentication bitstream diagram.

and then verify the previous owners according to the history iteratively until the OCM. In this way, the system developer can track the journey of the FPGA devices throughout the supply chain with the FPGA chain. If the verification passed, that is, successfully identifying the OCM, all involved entities can reach the consensus to update the current owner from the PCB assembler to the system developer (ownership history column in Figure 1d). Otherwise, the transaction will be rejected and items would be returned.

Bitstream registration

System developers compile their hardware designs to generate configuration bitstreams, making the FPGAs become functional systems. As bitstream encodes the functionality of FPGA implementations, it is critical to protect them from malicious manipulations to avoid aftermaths like denial-of-service and information leakage [2]. The bitstream registration (see Figure 2c) procedure records the plaintext bitstream on the FPGA chain that can be used as a reference for future verification. The system developer will send the registration request to the trusted OCM first. After the request approval, OCM will run the *CheckJD* contract to see whether the ECID from the system developer exists in the ledger or not. If yes, OCM will execute *Update_Asset* to bind the bitstream with the asset (FPGA) object, that is, the purple field in Figure 1d will be linked to the configuration data. Note that the plaintext bitstream is also stored in a private way to avoid being reverse engineered by untrusted entities [2]. One may claim that system developers can upload their bitstreams to a conventional OCM-centralized database waiving the needs of a blockchain infrastructure, which might be more manageable but violate the data privacy policies since OCMs are considered to be trusted only for their hardware devices instead of the soft intellectual property (IP) encoded in the plaintext bitstream belonging to system developers.

Entering the user domain

When entering the user domain, FPGAs can be sold either without bitstream (bare chips and FPGA boards) or with bitstream (FPGA-based systems). End-users always care about the device's authenticity and bitstream integrity, whereas they are assumed to be unqualified to perform sophisticated inspections. They can rely on the FPGA-chain platform for authentication instead as presented in Figure 2d where an end-user purchases an FPGA-based system from a system distributor. The FPGA device might be counterfeit while the alongside bitstream could be tampered with by a system distributor. The first steps are identical to the transactions between supply chain entities but a successful ECID-based verification does not necessarily indicate an *authentic* part and a *compliant* bitstream. Therefore, the end-user will send the authentication request via DApp and receive a partial bitstream to be downloaded in the FPGA. The partial bitstream will package the essential in-field information as a response packet to be sent back to the FPGA chain for verification.

As illustrated in Figure 2e, the authentication partial bitstream hardcodes a ciphertext $\mathbf{ENC}_{\mathrm{KA}}$ (K_H) and a finite-state machine (FSM)-obfuscated key K_B [12], where K_A AND K_B are a pair of asymmetric keys while K_H is the HMAC key. On the FPGA side, the partial reconfiguration engine in the static region will load the partial bitstream on the *Dynamic Region* through the internal configuration access port (ICAP) interface which can access and/or write the FPGA ON-chip configuration memory at run-time. Then, the authentication bitstream will access the read from the **DNA_Port**, collect PUF responses (R_i), and retrieve the configuration frames of the functional region through the ICAP FSM. K_B is unrolled and decrypts K_H from $\mathbf{ENC}_{\mathrm{KA}}$ (K_H) in parallel. The final response packet will cover the ECID, PUF response, $HMAC_{KH}$ (configuration data of the functional region), and HMAC over the three segments using K_H as well as the part marking (end-user input). The response packet will be checked against the reference information by the FPGA chain by following the protocol *Authenticate_Asset.* This protocol can enable secure authentication even considering the *man-in-the-middle* attack since K_H can only be decrypted by the obfuscated K_B, that is, only time-consuming full bitstream reverse engineering can extract it (state-of-the-art techniques require >10 minutes [2]). An authentication session is only valid within the time threshold (e.g., 3 minutes) and rejects late packets to ensure security. If *Authenticate_Asset* passed, the involved entities would achieve the consensus transferring the ownership from the system distributor to the end-user. Note that, for those without bitstream scenarios (e.g., bare chip or FPGA development boards), the authentication procedure is similar but without the need for checking the bitstream integrity (e.g., ICAP FSM is not required in the authentication bitstream in Figure 2e).

With the existing identification and intrinsic information, the FPGA chain can model the FPGA devices and bitstreams in the blockchain infrastructure to trust-worthily track and authenticate them thwarting targeted supply chain attacks in the previous subsection, whereas conventional solutions like ECID query and PUF cannot achieve the identical effectiveness individually as discussed in the previous section.

Evaluation and discussions

Security evaluation

As discussed in the previous section, there are five attack scenarios to be addressed by the FPGA chain. We illustrate the example scenarios against the attacks using the FPGA chain in Figure 3 and elaborate on them below.

- *Recycled devices:* Malicious distributors can import recycled devices from electronic recyclers (e.g., from offshore) and sell them as new ones to downstream entities such as PCB assemblers for more profits. Using recycled devices will undermine the reliability and security of the system significantly. The PCB assemblers as buyers (see Figure 3a) can resort to the FPGA-chain infrastructure to verify the device authenticity of such recycled chips by checking the current owner/stage of the device ECID; the FPGA chain might report *Verify_Assets* failure because the device already arrives at a very end phase (e.g., customers or recyclers) according to the ledger records while it has come back to life again (e.g., IC distribution).
- *Overproduced devices:* As illustrated in Figure 3b, a rogue foundry can fabricate more FPGA devices beyond the contract and rely on malicious distributors to sell them as legitimate parts. There are two possible circumstances, that is, an overproduced FPGA with an illegitimate or legitimate ECID. The former is easy to be detected by the FPGA chain since the device ECID is never enrolled and cannot be found in the ledger. The latter case can be detected according to the ownership history and database since the current owner of its legitimate counterpart can be someone else instead of the seller (e.g., malicious distributor) in this transaction.
- *Remarked devices:* Figure 3c depicts an example case of remarked devices where the malicious distributors modify the top marking of an FPGA

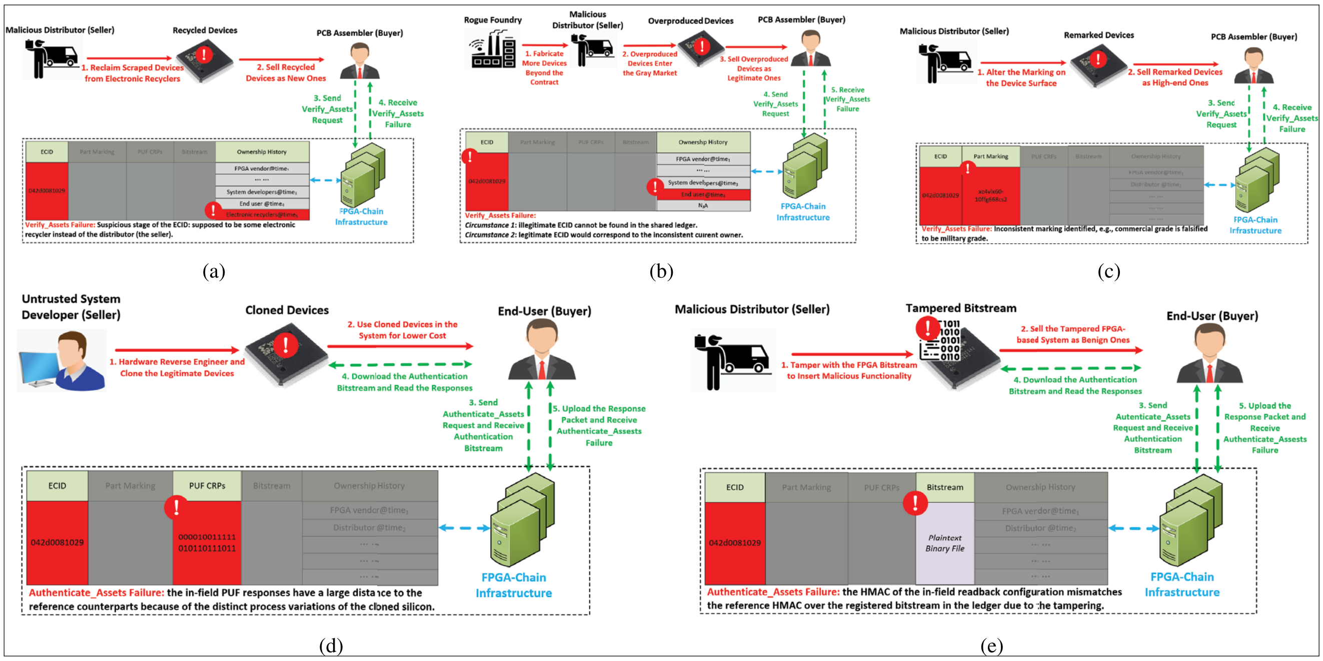

Malicious Distributor (Seller)
Recycled Devices
PCB Assembler (Buyer)
1. Reclaim Scraped Devices from Electronic Recyclers
2. Sell Recycled Devices as New Ones
3. Send Verify_Assets Request
4. Receive Verify_Assets Failure
FPGA-Chain Infrastructure
Verify_Assets Failure: Suspicious stage of the ECID: supposed to be some electronic recycler instead of the distributor (the seller).
(a)
Rogue Foundry
Malicious Distributor (Seller)
Overproduced Devices
PCB Assembler (Buyer)
1. Fabricate More Devices Beyond the Contract
2. Overproduced Devices Enter the Gray Market
3. Sell Overproduced Devices as Legitimate Ones
4. Send Verify_Assets Request
5. Receive Verify_Assets Failure
Verify_Assets Failure:
Circumstance 1: illegitimate ECID cannot be found in the shared ledger.
Circumstance 2: legitimate ECID would correspond to the inconsistent current owner.
(b)
Malicious Distributor (Seller)
Remarked Devices
PCB Assembler (Buyer)
1. Alter the Marking on the Device Surface
2. Sell Remarked Devices as High-end Ones
3. Send Verify_Assets Request
4. Receive Verify_Assets Failure
Verify_Assets Failure: Inconsistent marking identified, e.g., commercial grade is falsified to be military grade.
(c)
Untrusted System Developer (Seller)
Cloned Devices
End-User (Buyer)
1. Hardware Reverse Engineer and Clone the Legitimate Devices
2. Use Cloned Devices in the System for Lower Cost
4. Download the Authentication Bitstream and Read the Responses
3. Send Authenticate_Assets Request and Receive Authentication Bitstream
5. Upload the Response Packet and Receive Authenticate_Assets Failure
ECID
Part Marking
PUF CRPs
Bitstream
Ownership History
042d0081029
000010011111 010110111011
FPGA vendor@time1
Distributor @time2
FPGA-Chain Infrastructure
Authenticate_Assets Failure: the in-field PUF responses have a large distance to the reference counterparts because of the distinct process variations of the cloned silicon.
(d)
Malicious Distributor (Seller)
Tampered Bitstream
End-User (Buyer)
1. Tamper with the FPGA Bitstream to Insert Malicious Functionality
2. Sell the Tampered FPGA-based System as Benign Ones
4. Download the Authentication Bitstream and Read the Responses
3. Send Autenticate_Assets Request and Receive Authentication Bitstream
5. Upload the Response Packet and Receive Authenticate_Assets Failure
Plaintext Binary File
FPGA-Chain Infrastructure
Authenticate_Assets Failure: the HMAC of the in-field readback configuration mismatches the reference HMAC over the registered bitstream in the ledger due to the tampering.
(e)

Figure 3. FPGA chain thwarts typical supply chain attacks. (a) Recycled devices detection example. (b) Overproduced devices detection example. (c) Remarked devices detection example. (d) Cloned devices detection example. (e) Tampered bitstream detection example.

to a higher grade for a 10× or even 100× more profit space. With the FPGA chain, the marking information is also bonded with the ECID (asset object) in the ledger. Once an inconsistency between the marking from buyers and the reference marking belonging to the same FPGA is found, a remarked device is detected.

- *Cloned devices:* A system developer, albeit being trusted in terms of FPGA bitstream, might intend to create cloned devices for a reduced cost. Cloned devices are replicas of authentic ones and thus have the same ECIDs (see Figure 3d). Detecting a cloned device is similar to dealing with the overproduced chip with a legitimate ID. One additional case is when the cloned device flows faster than its authentic counterpart (e.g., the counterfeits enter the distribution network at first), the ECID-based identification would wrongly consider the replica as the authentic part. Nevertheless, before reaching the user domain, the authentication procedure as articulated in the previous section would provision a soft PUF primitive on the target FPGA collecting the in-field PUF responses which depend on the device's intrinsic process variations. In this way, the cloned copies can be differentiated if the in-field responses deviate much from their reference counterparts stored in the FPGA chain.

- *Tampered bitstream:* The configuration data might be tampered with by malicious distributors after bitstream integration for incorporating malicious circuitry (see Figure 3e). The FPGA chain can detect the bitstream corruptions through the authentication protocol that the FPGA configuration data of the functional regions will be accessed during the run-time to calculate the in-field HMAC accordingly. The mismatch between the in-field HMAC and the pre-computed value using the registered plaintext bitstream from the FPGA chain indicates a tampered bitstream. The HMAC key is random per session and well protected through hardware obfuscation and asymmetric cryptography, so the attacker cannot replay or modify the response packet arbitrarily.

Implementation and overhead evaluation

The FPGA chain has been implemented by the blockchain infrastructure, as detailed in [8]. The prototype infrastructure is Hyperledger fabric-like, consisting of seven peer nodes (each has an Intel Xeon processor and 32 GB RAM) standing for organizations such as OCM and IC distributors. The smart contracts are packaged on every individual node, while the DApp is developed with Node.js, while the computational performance is evaluated that we can register 1,000 devices in 20 min at a speed of 40 transactions per minute. Another important analysis is to evaluate the time and area overhead of the authentication protocol where the partial bitstream design (the dynamic region in Figure 2e) should be compact and efficient. The following statistics are reported on a mid-end FPGA platform Xilinx ZCU104. A soft 1,024-bit RSA design was used to enable asymmetric cryptographic operations (652 slices). The ECID reader was essentially a shifter register connecting to the **DNA_PORT** primitive to access the ECID (28 slices). Besides, we selected an area-efficient and stable 16-bit RO PUF scheme for enrolment and authentication purposes (213 slices). The largest module was the soft HMAC along with the ICAP FSM occupying 1,985 slices in total. The overall resource utilization is around 3.21% of the entire FPGA fabric. Given the variety of FPGA fabric, we evaluate the area overhead on Xilinx VC709 and Synopsys HAPS DX7 platforms as well which are roughly 2.74% and 2.09%, respectively, in terms of slices. As for the time overhead covering the network transmission of the partial bitstream and the response packet as well as the statistics generation (e.g., HMAC calculation), the entire procedure was around 1.13 s (the main clock was assumed to be 100 MHz). Therefore, the FPGA implementation for in-field authentication is ideal for verifying the authenticity of the FPGA-based system because of the small overhead.

The security threats in the convoluted FPGA supply chain are longstanding concerns that have not yet been addressed holistically. We propose our FPGA-chain framework to deploy a consortium blockchain infrastructure featuring fair data management, flexible smart contracts, and secure authentication protocols to provide substantial protection to the entire FPGA supply chain against counterfeit devices including recycling, cloning, overproduction, and remarking as well as bitstream tampering threats. The comprehensive evaluation demonstrates the effectiveness and acceptable overhead of our FPGA chain. We will further enhance the FPGA chain's framework by introducing active IP metering and compatibility with emerging devices. ■

References

[1] *FPGA Market Report*. Accessed Nov. 28, 2020. [Online]. Available: https://www.grandviewresearch.com/industry-analysis/fpga-market

[2] T. Zhang et al., "A comprehensive FPGA reverse engineering tool-chain: From bitstream to RTL code," *IEEE Access*, vol. 7, pp. 38379–38389, 2019.

[3] U. Guin et al., "Counterfeit integrated circuits: A rising threat in the global semiconductor supply chain," *Proc. IEEE*, vol. 102, no. 8, pp. 1207–1228, Jul. 2014.

[4] R. S. Chakraborty et al., "Hardware Trojan insertion by direct modification of FPGA configuration bitstream," *IEEE Design Test*, vol. 30, no. 2, pp. 45–54, Apr. 2013.

[5] X. Xu et al., "Electronics supply chain integrity enabled by blockchain," *ACM Trans. Design Autom. Electron. Syst.*, vol. 24, no. 3, pp. 1–25, Jun. 2019.

[6] B. Olney and R. Karam, "Tunable FPGA bitstream obfuscation with Boolean satisfiability attack countermeasure," *ACM Trans. Design Autom. Electron. Syst.*, vol. 25, no. 2, pp. 1–22, Mar. 2020.

[7] M. T. Rahman et al., "The key is left under the mat: On the inappropriate security assumption of logic locking schemes," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, Dec. 2020, pp. 262–272.

[8] N. Vashistha et al., "EChain: A blockchain-enabled ecosystem for electronic device authenticity verification," *IEEE Trans. Consum. Electron.*, vol. 68, no. 1, pp. 23–37, Feb. 2022.

[9] M. Ender, A. Moradi, and C. Paar, "The unpatchable silicon: A full break of the bitstream encryption of Xilinx 7-series FPGAs," in *Proc. 29th USENIX Security Symp.*, 2020, pp. 1803–1819.

[10] M. N. Islam and S. Kundu, "Enabling IC traceability via blockchain pegged to embedded PUF," *ACM Trans. Design Autom. Electron. Syst.*, vol. 24, no. 3, pp. 1–23, Jun. 2019.

[11] J. Zhang et al., "A PUF-FSM binding scheme for FPGA IP protection and pay-per-device licensing," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 6, pp. 1137–1150, Jun. 2015.

[12] M. Hoffmann and C. Paar, "Stealthy opaque predicates in hardware—Obfuscating constant expressions at negligible overhead," 2019, *arXiv:1910.00949*.

Tao Zhang is pursuing a PhD at the Electrical and Computer Engineering Department, University of Florida, Gainesville, FL 32611 USA. His research interests include hardware security and trust, supply chain security, and FPGA security.

Fahim Rahman is a research assistant professor at the Electrical and Computer Engineering Department, University of Florida, Gainesville, FL 32611 USA. His research interests include microelectronics security and assurance.

Mark Tehranipoor is the Intel Charles E. Young Preeminence Endowed Chair Professor in Cybersecurity at the Electrical and Computer Engineering Department, University of Florida, Gainesville, FL 32611 USA. His research interests include microelectronics security and trust as well as semiconductor supply chain security.

Farimah Farahmandi is an assistant professor at the Electrical and Computer Engineering Department, University of Florida, Gainesville, FL 32611 USA. Her research interests include hardware security and formal verification.

■ Direct questions and comments about this article to Tao Zhang, Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611 USA; tao.zhang@ufl.edu.

Conference Report

The 2022 International Conference on Computer-Aided Design (ICCAD)

Tulika Mitra
School of Computing
National University of Singapore
Singapore 119077

Jointly sponsored by ACM and IEEE, the International Conference on Computer-Aided Design (ICCAD) is a premier forum to explore new challenges, present leading-edge innovative solutions, and identify emerging technologies in the electronic design automation (EDA) research areas. ICCAD covers the full range of computer-aided design (CAD) topics—from the device and circuit levels up through the system level, as well as post-CMOS design.

After two years of virtual conferences due to the global COVID-19 pandemic, we were excited to organize the 41st edition of ICCAD as a hybrid conference from 30 October to 4 November 2022, in San Diego, CA, USA. The conference was primarily an in-person event enabling personal interactions and extensive networking, aspects that cannot be replicated in virtual platforms. At the same time, we offered virtual sessions for the speakers and the attendees who cannot attend in person due to travel restrictions or health issues. The virtual sessions were held fully online in the morning followed by the in-person sessions. Approximately 40 sessions were held in person, while 15 sessions were virtual. Prerecorded video presentations of all the talks, irrespective of whether they are part of a virtual or in-person session, were available on the virtual platform for all the attendees. The participants could discuss the papers with the authors through the virtual platform. The conference was immensely successful with a record 533 registered participants including 336 onsite attendees.

Digital Object Identifier 10.1109/MDAT.2023.3238340
Date of current version: 10 March 2023.

ICCAD has a long-standing tradition of producing cutting-edge, innovative technical programs for attendees. Following the tradition, the members of the executive committee, the technical program committee, and numerous volunteers spent an enormous effort to prepare an outstanding technical program for this year as well. We are glad to announce that we again had a significant increase in the number of regular paper submissions with a record 595 papers going through the complete peer-review process. This strong submission record amidst a global pandemic emphasizes the high relevance and recognition of the conference within, but not limited to, the CAD community. For handling such an enormous submission number, we carefully created 15 tracks and invited 203 outstanding technical program committee members from both industry and academia worldwide for these tracks. The TPC meeting was conducted as an online event without compromising the quality of the double-blind review process. Finally, the program committee has selected 132 papers with an acceptance rate of 22% and yielded 43 regular sessions on diverse topics. In addition, we had ten special sessions and two embedded tutorials on topics that complement the regular sessions.

We were delighted to present three distinguished keynote speakers: the Monday morning keynote on Democratizing IC Design and Customized Computing was given by Prof. Jason Cong from the University of California, Los Angeles. On Tuesday, Prof. Farinaz Koushanfar from the University of California, San Diego, presented the IEEE CEDA Luncheon Distinguished Lecture on Automated Cryptographically-Secure Private Computing. Shankar Krishnamoorthy from Synopsys presented the Wednesday keynote on Atoms to Silicon to Systems Hyper-Convergence to realize the next wave of semiconductor innovations. Finally, Prof. Rob Rutenbar presented the ACM SIGDA Pioneering Achievement Award speech on Tuesday. As expected, these keynotes and speeches were inspiring, insightful, and informative.

We had six interesting workshops on Thursday and one on Friday covering various new and established topics.

Some of these workshops [Workshop on Accelerator Computer-Aided Design (ACCAD), Workshop on Hardware and Algorithms for Learning On-a-Chip (HALO), Workshop on Open-Source EDA Technology (WOSET), and Top Picks in Hardware and Embedded Security] are long-time staples of ICCAD, while the Workshop on Zero Trust Hardware Architectures tests the waters for the first time. Two of these workshops are colocated with ICCAD: 24th ACM/IEEE International Workshop on System-Level Interconnect Pathfinding (SLIP) and Sustainable Hardware Security (SUSHI). The latter in a new interactive workshop that brought together around 50 invited experts from academia, industry, and government to explore sustainable security for computing platforms.

Last but not the least, ICCAD hosted several contests and competitions. The *CAD Contest* is a multimonth, team-based research and development competition, focusing on advanced, real-world problems in the field of EDA. The *CADathlon* is an all-day programming competition focusing on practical problems at the forefront of CAD and EDA. The *ACM Student Research Competition (SRC)* at ICCAD provides an opportunity for undergraduate and graduate students to share research results and exchange ideas with other students, judges, and conference attendees. This year, we also debuted the ACM/IEEE *TinyML Design Contest*—a team-based, multimonth, research and development competition, focusing on real-world problems that require the implementation of machine-learning algorithms on low-end microprocessors/microcontrollers. Finally, ICCAD 2022 also featured the first-ever *Job Fair* where students and professionals met with representatives from approximately ten industry and research organizations.

ICCAD aims relentlessly at being the ultimate destination for cutting-edge EDA research and emerging CAD technologies. The organization of ICCAD is only possible with continuous support and help from the sponsors and many volunteers: the program chair with the program committee members, the organizers of the workshops, contests, and job fairs, and all members of the organization committee. We are grateful for their commitment and dedicated contributions as well as the attendance, interaction, and support of the community in shaping this year's hybrid event into a memorable one.

The next ICCAD will take place in San Francisco, CA, USA, between 29 October and 2 November 2023. Please follow iccad.com for more details and updates. ■

Tulika Mitra is a vice-provost (academic affairs) and a provost's chair professor of computer science at the National University of Singapore, Singapore 119077. Her research focuses on the hardware–software codesign of smart, energy-efficient, and safety-critical embedded computing systems. Mitra has a master's from the Indian Institute of Science, Bengaluru, India, and a PhD from Stony Brook University, Stony Brook, NY, USA. She has served/is serving as a member of the ACM Publications Board.

■ Direct questions and comments about this article to Tulika Mitra, School of Computing, National University of Singapore, Singapore 119077; tulika@comp.nus.edu.sg.

Is There an Answer?

Scott Davidson

Back when I first got into the electronic test, over 40 years ago, things were simple. You applied a test to a chip, usually a functional test, and you got a result. The expected value, obtained during simulation, was a series of vectors, ones, and zeroes, which was simple to compare with the circuit output. If your simulation and test program were good, the outputs would match except when the circuit was faulty.

We soon moved to structural testing, where you mostly cared about the outputs of the scan chains. A bit trickier, but still not a big problem.

Now, based on several articles in this issue of *IEEE Design&Test*, things are more complicated. Defects do not necessarily cause a system to produce incorrect results. A structural test of AI hardware is not a problem, but I am not sure how to do a functional test.

For most of the world, testing has nothing to do with digital circuits. Reading this issue made me think of the more normal kind of testing. Let us consider the tests we took (or gave) in college.

In a way, a multiple-choice or short-answer test can be considered a structural test. Each question tests for a specific item of knowledge. If a student memorized the right set of facts, they might do very well on these tests, even if they did not understand the subject at all. An essay question is more of a functional test. Properly written, it can test if a student understands how the parts of a topic area fit together. Now, often when we grade an essay test, we will have a list of things we expect the student to cover. That makes the functional essay test a bit more like the structural short-answer test. Structural tests are easier to grade, just like electronic testing. Grading of multiple-choice tests like the SAT is automated, but grading of essay tests is not. Yet.

Which brings us back to AI. As I write, the news is full of panic about students using ChatGPT to do their homework, specifically to write essays. I asked it to write this column, but it said “Sorry, Davidson, I can’t do that,” so be assured that this column is being written the old-fashioned way—on a computer. But it appears that this tool does as good or better a job as many students. I am not sure what to think about that.

If students are using AI to write essays, it seems only fair to allow professors to use AI to grade essays. I wonder if ChatGPT can do that. I wonder if anyone has tried it.

This reminds me of a running gag in the excellent movie *Real Genius,* which is set at a university that resembles CalTech. A large lecture class is being taught by a boring professor. A student brings a tape recorder, puts it on their seat, and leaves. As the movie progresses, more and more seats are occupied by tape recorders, until the professor tapes his lecture and we see a tape recorder lecturing in a room filled with other tape recorders. Maybe someday AIs will both take and grade tests, and students and teachers can sit outside on the lawn and learn that way. After all, I doubt Plato gave Aristotle multiple-choice tests. ■

Direct questions and comments about this department to Scott Davidson; davidson.scott687@gmail.com.

Digital Object Identifier 10.1109/MDAT.2023.3241567
Date of current version: 10 March 2023.