IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received 27 March 2024, accepted 11 April 2024, date of publication 22 April 2024, date of current version 29 April 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3392240

RESEARCH ARTICLE

FPGA Implementation of a Fault-Tolerant Fused and Branched CNN Accelerator With Reconfigurable Capabilities

RIZWAN TARIQ SYED[1], **YANHUA ZHAO**[1], **JUNCHAO CHEN**[1], **MARKO ANDJELKOVIC**[1], **MARKUS ULBRICHT**[1], **AND MILOS KRSTIC**[1,2]

[1]IHP—Leibniz-Institut für Innovative Mikroelektronik, 15236 Frankfurt (Oder), Germany
[2]Chair of Design and Test Methodology, University of Potsdam, 14469 Potsdam, Germany

Corresponding author: Rizwan Tariq Syed (syed@ihp-microelectronics.com)

This work was supported by the Federal Ministry of Education and Research of Germany in the project "Open6GHub" under Grant 16KISK009.

ABSTRACT The ImageNet moment was a turning point for Convolutional Neural Networks (CNNs), as it demonstrated their potential to revolutionize computer vision tasks. This triumph of CNNs has motivated solving even more complex problems involving multiple tasks from multiple data modalities. Conventionally, a single CNN accelerator has been optimized to perform just one task or multiple correlated tasks. This study presents a shared-layers approach that leverages the pattern-learning capabilities of CNNs to perform multiple uncorrelated tasks from different modalities using a single hardware accelerator. We overcame the challenge of data imbalance in multi-modal learning by synthetic data generation. We achieved an average classification accuracy above 90% on a single CNN accelerator, which would otherwise require three accelerators. Due to the reliability concerns imposed by transistor shrinking and aging, we extended the shared layers methodology and introduced a fault-tolerant CNN accelerator with reconfigurable capabilities supporting fault-tolerant (FT), high-performance (HP), and de-stress (DS) modes. FT mode provides high reliability against soft errors utilizing double/triple modular redundancy, HP mode offers peak performance of 0.979 TOPs using parallel execution, and DS mode reduces dynamic power consumption by up to 68.6% in clock-gated design and even more using a partial reconfiguration method, contributing to decelerating the aging process of the circuit. We have comprehensively evaluated two different CNN architectures (i.e., fused and branched), for three distinct tasks, in three different operating modes, based on accuracy, quantization, pruning, hardware resource utilization, power, energy, performance, and reliability.

INDEX TERMS Multi-task learning, multi-modal learning, convolutional neural network, reliability, FPGAs, reconfigurability.

I. INTRODUCTION

CNNs have transformed many applications, ranging from self-driving cars and smart video surveillance to intelligent manufacturing and medical imaging. With the increasing number of sensor modalities, CNNs are growing in complexity as they evolve from processing images/patterns captured by a single image sensor for basic object detection to handling multiple data streams from multiple sensors, performing classification tasks across a range of diverse objectives.

The challenge at hand is that running deep CNN models is a resource-intensive process, and deploying these models, with millions of parameters, on edge devices has become an increasingly pressing matter. CNNs, implemented on edge devices (i.e., ASICs, FPGAs) as hardware accelerators, offer benefits in terms of lower latency and enhanced data security and privacy. However, due to the varying AI requirements and workload, hardware resource utilization quickly reaches a boundary. An increase in hardware

The associate editor coordinating the review of this manuscript and approving it for publication was Ilaria De Munari.

resource utilization leads to higher power consumption. To address these concerns, several model compression methods (i.e., pruning, quantization, knowledge distillation, low-rank factorization, etc.) have been proposed. Model compression aims to reduce the model size so that it can be deployed on low-power and resource-constraint devices without significant accuracy degradation. In this context, we aspire to provide a fresh perspective on the issue of saving hardware resources and reducing power consumption using 'Shared Layers' approach. This is discussed in our earlier work [1], where we leverage the fundamental capability of CNNs of learning to recognize patterns and train multiple distinct tasks from different modalities, thereby forcing one application-specific CNN accelerator to learn the common features between the tasks, which would otherwise require three separate accelerators. We have evaluated fused and branched CNN models with different model compression methods. We conclude that the shared layers approach, assisted by pruning and quantization methods, substantially reduces hardware resources and power.

The second challenge arises concerning the reliability of the hardware components (i.e., ASICs or FPGAs) utilized for the CNN accelerator, especially given the ubiquity of CNNs in diverse applications, ranging from healthcare and entertainment to manufacturing and safety-critical applications, etc. There are high-reliability standards in safety-critical applications (e.g., self-driving cars, space, etc.) because the consequences of a failure can be catastrophic and lead to loss of life or property damage.

Over the years, we've witnessed a rise in hardware performance within microprocessors and hardware accelerators, driven by transistor size shrinking to the Very Deep Sub-Micron (VDSM) level. The downsizing of transistors results in increased susceptibility to transient faults due to reduced threshold voltages and narrower noise margins [2]. In addition to this, transistor aging presents another substantial reliability concern due to the gradual deterioration of circuit performance over time due to Hot Carrier Injection (HCI) and Bias Temperature Instability (BTI) [32], [33]. Hence, to ensure fault-free CNN inference on hardware, it is essential to have a rigorously validated fault-tolerant and aging-aware methodology in place.

Therefore, with added hardware redundancy, we further extend the concept of shared layers to propose reconfigurable CNNs. The primary idea is that CNN accelerators should adapt based on the changing needs of accuracy, power, latency, reliability, etc. Thus, our reconfigurable accelerator performs multiple tasks in 1) Fault tolerant (FT), 2) De-stress (DS), and 3) High performance (HP) modes. Executing multiple tasks in various operating modes on CNN accelerators positioned this study uniquely, which was otherwise observed most commonly in microprocessors. Noteworthy contributions of the paper are:

(1) Fault-tolerant multi-modal CNN accelerators based on shared-layers methodology, with reconfigurable capabilities qualified to operate in three distinct modes: 1) FT, 2) HP, and 3) DS.
(2) Comprehensive experimental results of three operating modes, which include:
    (a) Fault analysis on various fault models (Single Event Transients (SETs), Single Event Upsets (SEUs), Multi-bit upsets (MBUs), SEU in FPGA configuration memory) in FT mode.
    (b) Examination of latency and energy consumption in the HP mode.
    (c) Comparative assessment between clock gating (CG) and partial reconfiguration(PR) methods to reduce the dynamic power consumption in the DS mode.

The subsequent sections of the paper are structured as follows. Section II distinguishes our work from state-of-the-art. Section III delves into the background of CNNs and multi-modal multi-task learning (MMMT). Section IV explains the fused and branched CNN models based on the shared-layers approach, while section V and VI discuss its workflow and experimental results, respectively. We further develop the idea of shared layers and introduce fault-tolerant reconfigurable accelerators in section VII. Section VIII concludes this study.

II. RELATED WORK

In order to address the evolving challenges posed by changing AI application requirements, several research groups have proposed the idea of reconfigurable or adaptive accelerators. The central concept revolves around AI accelerators being adaptable to changing requirements such as accuracy, power efficiency, latency, reliability, etc. Thus, diverse notions related to reconfigurable deep neural networks (DNNs) have been introduced.

Authors in [3] have different bitstream configurations for different DNN models with varying quantization levels for a video processing application. They dynamically change the bitstream of the DNN model for a tradeoff between accuracy and power. Samples per second is one of the significant advantages when transitioning between various FPGA bitstreams. However, in the context of reconfigurability, [4] and [7] perform re-programming weights to support different DNN architectures and unique data compression algorithms, respectively. Reference [7] also evaluates the impact of faults in DNNs. Reference [5] has taken a different approach and suggests an energy-efficient reconfigurable CNN accelerator design using streaming data flow architecture. This study decomposes large CNN kernel computations into small kernel-sized computations and improves energy efficiency by reducing nonessential data movement. The study [6] proposes a reconfigurable binary neural network accelerator that conducts adaptive loading and processing of data in CNNs kernels to gain performance increase. While [8] suggests a configurable architecture for implementing CNNs, which supports hybrid quantization (i.e., different bit widths for different layers) with the primary objective of increasing

TABLE 1. Comparison with the state-of-the-art.

	Reconfigurability Type	Fault Tolerance (soft errors, permanent faults, etc.)	High Performance Support	Multi-Task	Multi -Modal	Fault Models
[26], [10]	x	x	yes	x	x	x
[11]	bitstreams reconfiguration	yes (aging)	x	x	x	permanent faults
[22], [23], [12] [13], [14], [24]	x	yes (radiation)	x	x	x	SEU
[15]	x	yes (manufacturing defects)	x	x	x	permanent faults
[16]	x	yes	x	x	x	permanent faults
[3]	bitstreams reconfiguration	x	yes	x	x	x
[17]	x	x	yes	yes	x	x
[4]	weights re-programming	x	x	x	x	x
[18]	x	x	yes	x	yes	x
[37]	voltage and frequency scaling	x	yes	x	x	x
[7]	weights re-programming	yes (radiation)	yes	x	x	SEU
[5]	large kernel decomposition	x	yes	x	x	x
[6]	adaptive loading and processing data	x	yes	x	x	x
[8]	hybrid quantization support	x	yes	x	x	x
[9]	data path reconfiguration	x	yes	x	x	x
Proposed	supporting multiple operating modes (FT, DS, HP)	yes(radiation, aging)	yes	yes	yes	SEU, SET, MBU, SEUs in CRAM

TABLE 2. Comparison with previous fault-tolerant DNN studies.

	This Study	[7]	[15]	[22]	[23]	[12]	[13]	[14]	[24]	[16]
Target Hardware	VCU118	ASICs 65nm	ASICs, OSU FreePDK 45 nm	x	x	x	x	x	x	ZCU104
Model	6 Layer CNN	autoencoder	MLP, AlexNet	VGG16	AlexNet, ResNet18, ResNet50	CovNet, AlexNet	VGG16, ResNet50, InceptionV3	YOLO, Faster R-CNN	LeNet, VGG16	9 layer CNN
Model Compression	pruning quantization	quantization	x	pruning	pruning quantization	quantization	x	x	pruning quantization	quantization
Fault Model	SEU, SET, MBU, SEUs in CRAM	SEU	stuck-at	SEU	SEU	SEU	SEU	SEU	SEU	stuck-at
FI Location	complete accelerator architecture	x	faults in the data-path (Parameters maps to MAC unit)	CNN parameters	CNN parameters	FI data-path of accelerators in C/C++ based DNN simulator	CNN Parameters	bit flips in memory	CNN weights	output feature maps of CNN layers
Fault Injection Level	RTL, FPGA configuration memory	x	gate- level netlist	software (Pytorch based)	software (Pytorch based)	Software (DNN simulator in C++)	software (Tensorflow based)	software (Python based)	software (Pytorch based)	FPGA-accelerated error injection
FT Methodology	shared layers-based TMR/DMR with reconfigurability	TMR	FAP: fault-aware pruning FAP+T: fault-aware pruning and retraining	utilize reliability analysis to perform selective redundancy	utilize model compression to increase reliability	SED: Symptom-based Error Detectors SLH: Selective Latch Hardening	x	spatio-temporal correlation	virtual and physical redundancy	selective channel replication and fault-aware scheduling of PEs

the performance with little accuracy loss. The investigation by [9] recommends performing data path reconfiguration to promote data reuse patterns, which reduces total energy for different CNN models. This study introduces a layer-based scheduling framework to balance power efficiency and performance for different convolutional neural network models. The quantitative comparison in terms of hardware resource utilization, performance, power, energy, operating frequency, model compression methods, target hardware, and a number of sensor modalities and tasks performed in section VII D.

Reliability is a primary consideration when deploying DNN models for safety-critical applications. Therefore, numerous investigations have been conducted on fault-tolerant deep neural networks. The study [15] examines a DNN accelerator utilizing a systolic array architecture (i.e., Google Tensor Processing Unit) and suggests two fault-tolerant methodologies, fault-aware pruning (FAP) and fault-aware pruning + retraining (FAP+T). In FAP, pruning is leveraged to bypass the fault MACs, which are causing accuracy degradation. In FAP+T, an additional on-device retraining step is added to recover the accuracy loss caused by the missing MAC units. Authors in [22] have performed a reliability evaluation of pruned DNNs and recommended adding selective redundancy on vulnerable parts of the DNNs. A similar analysis was performed by [23], where authors suggested model compression (i.e., pruning, quantization) as an impressive way to improve the reliability of DNNS. The study [12] has proposed two FT methodologies, i.e., symptom-based error detectors (SED) and selective latch hardening (SLH). The former method monitors the output of DNN's activation functions, while the latter method adds redundancy in the vulnerable parts of the hardware. Authors in [7] suggest a triple modular redundant reconfigurable DNN accelerator for data compression using an autoencoder model, while authors in [13] have performed reliability analysis of VGG16, ResNet50, and InceptionV3 against transient faults happening in the model's parameter and summarized that the nature of the layers (batch normalization layers, use of shortcut connections) can also play a part in the resiliency of the CNN model. The study [14] proposed an inter-frame spatiotemporal correlation method to detect errors in CNNs, using the input and output correlation information while processing multiple frames. Reference [24] has done a resiliency analysis on compressed DNNs' to come up with virtual and physical redundancy to mitigate errors. Reference [16] have performed FPGA-accelerated error injection on output feature maps of CNN layers and have presented selective channel replication and fault-aware scheduling of PEs to mitigate the impact of faults. Other fault-tolerant methods include ensemble learning-based robustness [53], Knowledge distillation-based redundancy [54], algorithm-based fault tolerance (ABFT) [55], [56], [57], clipped activations function [50], [51] [52], and using arithmetic error codes to mitigate the reliability of the DNNs [58].

Table 1 provides a conceptual comparison of our approach to state-of-the-art studies, while Table 2 compares our work with selected studies related to FT methods in DNNs. Thus, we can summarize that,

- Several studies on fault-tolerant DNNs seem to overlook adaptivity/reconfigurability. Meanwhile, studies related to adaptive/reconfigurable DNNs primarily focus on increasing their performance. Thus, our study has addressed both aspects of reconfigurability in DNNs. i.e., high performance and fault tolerance.
- The discussions about hardware accelerators for multiple tasks and sensor modalities are largely ignored in the majority of FT DNN studies.
- Many studies have performed reliability analysis of DNNs, but most of this analysis is based on fault injection (FI) in neural network parameters (weights and biases) at the software level. Neural networks are eventually deployed on hardware (i.e., ASICs or FPGAs), and the hardware architecture consists of many components (i.e., memories, DSP blocks, state machines control, shared buffers, etc). Injecting faults only in the weights may not give complete insights into the vulnerability of the DNN accelerator. Therefore, our investigation concentrates on FI in the complete accelerator architecture.
- Many studies have only considered one task or sensor modality or performed fault injection analysis for one fault model. In contrast, this study has evaluated four fault models and performed FI analysis at the RTL level and in the configuration memory of FPGA.

This work proposes a distinctive reconfigurability framework employing the redundancy-assisted shared layers method to the CNN accelerator. To the best of our knowledge, this approach to the fault-tolerant CNN accelerator with reconfigurable capabilities, which aims for optimization of hardware resources, power consumption, high performance, and reliability for multiple tasks, has not been addressed.

III. MULTI-MODAL MULTI-TASK CNN MODELS

CNNs, a specialized type of neural network, have proven highly effective in image classification tasks. CNNs have the ability to develop an internal representation of an image or pattern. This allows the CNN model to learn the position and scale-invariant structures in the image data, which is very important when working with images. Due to the extensive use of CNNs in vision applications (i.e., image recognition, medical imaging, etc.), classification problems have also grown in complexity. CNNs have demonstrated exceptional performance, from having simpler CNN networks to solve one classification task (i.e., multi-class classification) to more extensive and complex CNN models solving classification problems from multiple modalities and tasks.

A. MULTI-MODAL LEARNING

Human beings perceive and interpret the world through multiple senses (i.e., multi-modal fashion), such as vision, hearing, touch, taste, and smell. Learning and processing information from multiple modalities (i.e., sensory organs) is essential for humans to interact with and understand the real world. Similarly, multi-modal learning (MML) involves processing data from multiple modalities, such as images, audio, text, and sensor data, to extract useful information and perform classification or prediction tasks. By processing data from multiple modalities, multi-modal CNNs can learn to recognize complex patterns and make predictions based on available information. The use of multiple sensors across various applications is increasing, resulting in an increased volume of collected data. The amalgamation of data samples from multiple modalities (such as camera, radar, lidar, and microphone) can be employed through sensor fusion to attain a more comprehensive and accurate understanding of the surrounding. CNNs, when trained on data from multiple modalities, learn the shared representation using various fusion methods (early fusion, late fusion, etc.). This approach can be effective in improving classification results by leveraging the strengths of the multiple modalities.

B. MULTI-TASK LEARNING

Depending on the application and data modality, a CNN model can perform a single task, such as identifying the type of clothing, or multiple tasks, such as identifying both the type and color of clothing. In machine learning, this is termed as multi-task learning (MTL). MTL is a learning paradigm where a model is trained to perform multiple related tasks simultaneously. An MTL model is trained for multiple tasks jointly by optimizing multiple loss functions [38]. The joint training helps the model learn the shared representation, reducing overfitting and facilitating the model to learn a more robust representation of the input data. MTL can be defined as multi-modal multi-task learning (MMMT) when the model can learn a shared representation of inputs across multiple data modalities and tasks, e.g., in a self-driving car scenario, an input data, which is captured by various sensors modalities (cameras, radar, lidar), gets processed by the AI model to detect multiple objects (i.e., pedestrians, lane markings, stop signs, etc.)

IV. PROPOSED SHARED LAYERS METHODOLOGY

The CNN models used in various applications are composed of a significant number of parameters, ranging from several thousand to millions. Ultimately, these models will be deployed on edge devices equipped with ASICs or FPGAs. These edge devices typically have limited computation resources and power budgets. Thus, deploying such models on edge devices poses a challenge due to the considerably high computational and energy requirements. This indicates a growing need to develop efficient techniques to minimize hardware resource utilization, power consumption, and

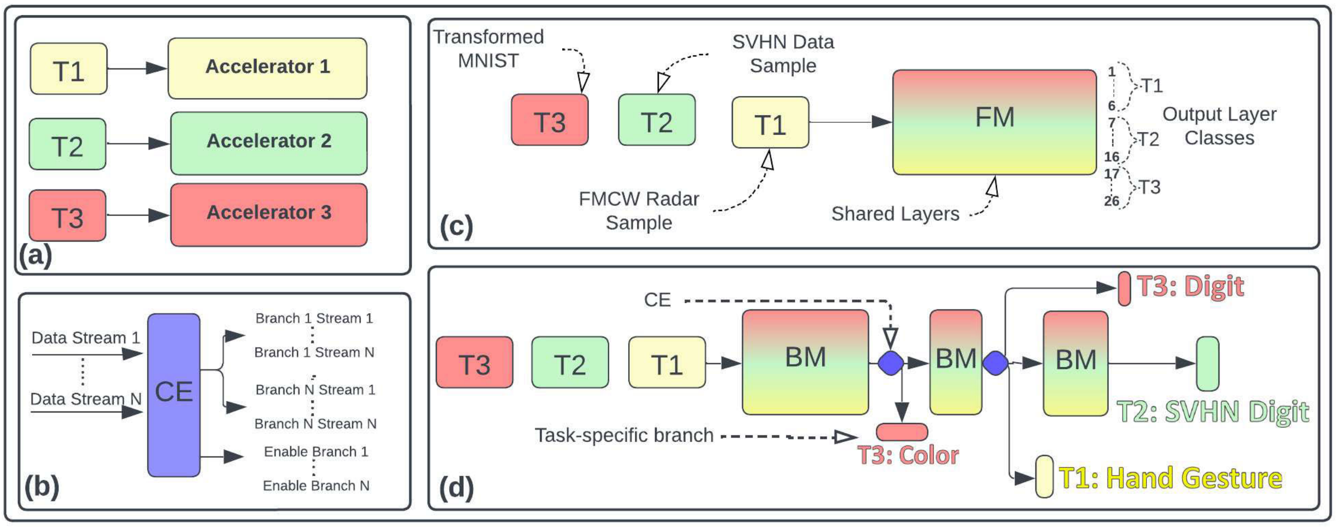


FIGURE 1. (a) Tasks execution on application-specific accelerators (b) Control element (c) Tasks execution on FM (d) Tasks execution on BM.

latency while meeting specific application requirements. The majority of studies on CNNs in the context of MTL and MMMT have focused on highly correlated tasks. Our approach provides a different perspective on MTL and MMMT, specifically regarding uncorrelated tasks. This perspective capitalizes on the core principle of how CNNs learn throughout their training process. The lower-level features of most of the images are the same. Lower layers learn the low-level features (i.e., edges, curves, blobs, etc.). The deeper we go in the network, the layers start to learn the high-level (or more abstract) features [39]. We leverage this concept to force one model to learn the lower-level features of multiple uncorrelated tasks. These uncorrelated tasks, characterized by matching low-level features, utilize the same layers. Once mapped to the hardware accelerator, these shared layers would mean sharing computing resources, leading to substantial savings in hardware resources and decreased power consumption. Our approach further facilitates the reuse of the weights for multiple tasks by avoiding unnecessary DRAM access, thereby leading to additional energy savings (e.g., Each DRAM access in a typical processing platform for CNNs consumes a significant amount of energy compared to the actual Multiply-Accumulate (MAC) operation [40]). Through this approach, a single application-specific accelerator can perform multiple tasks (Fig. 1(c)(d)), as opposed to the traditional way of utilizing three distinct accelerators (Fig. 1(a)). In section VII, the shared layers approach is further capitalized to re-configure the FPGA-based CNN accelerator during runtime to execute multiple tasks in FT, HP, and DS modes.

The effectiveness of our approach for multiple tasks is demonstrated through the creation of two types of CNN models: fused and branched. In the fused model (FM), all the tasks share all the layers of the neural network (Fig. 1(c)). In contrast, the branched model (BM) consists of tasks-specific branches and shares only particular layers (Fig. 1(d)). A mux-based logic is used to feed different data samples (i.e., corresponding to different tasks) to FM and BM in a TDM fashion. Although FM is comparatively easier to train and consumes slightly fewer hardware and power resources, it may not achieve the desired accuracy for a specific task. The BM involves a multi-stage training process and consumes slightly higher hardware and power resources than FM. However, BM offers several advantages, i.e., 1) It is slightly more resilient against faults, and it offers the possibility of task isolation in the presence of faults, 2) Provides task-specific bit-stream reconfiguration in FPGAs, 3) Introduces selective replication of only specific layers (e.g., more vulnerable layers or tasks-specific layers), 4) Enables sub-tasks to support MTL, 5) Provides extra layers to achieve more accuracy for specific tasks.

V. WORKFLOW FOR MULTIPLE TASKS EXECUTION ON FUSED AND BRANCHED MODELS

In this study, different tasks are emulated as different datasets, i.e., execution of a single task would mean performing a classification task on a single dataset. Thus, for three tasks, we have considered three datasets (i.e., radar samples for hand gesture detection [46], SVHN [19], MNIST [20]) from two different modalities (i.e., radar and camera image). These three tasks are illustrated in Fig. 3(a)(b)(c) and are executed on one application-specific CNN accelerator in a time-division multiplexing (TDM) manner.

Fig. 2 illustrates the overview of a comprehensive workflow which encompasses the following key stages: 1) Data preprocessing, 2) Model creation, training, and testing, 3) Utilizing the hls4ml framework to convert the Python-based CNN model to its HLS-compatible c/c++ equivalent. 4) This step involves conducting RTL and post-synthesis simulations to calculate latency and power (discussed in section VI). This step is further extended to perform fault injection analysis, to be covered in section VII A.

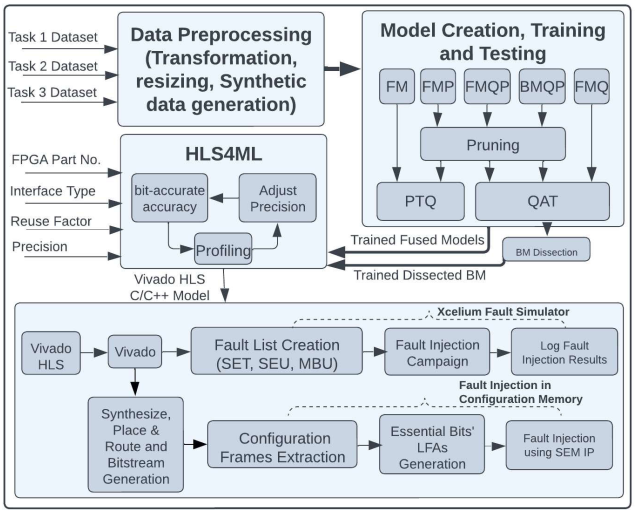

Task 1 Dataset
Task 2 Dataset
Task 3 Dataset
Data Preprocessing (Transformation, resizing, Synthetic data generation)
Model Creation, Training and Testing
FM
FMP
FMQP
BMQP
FMQ
Pruning
PTQ
QAT
FPGA Part No.
Interface Type
Reuse Factor
Precision
HLS4ML
bit-accurate accuracy
Adjust Precision
Profiling
Trained Fused Models
BM Dissection
Trained Dissected BM
Vivado HLS C/C++ Model
Xcelium Fault Simulator
Vivado HLS
Vivado
Fault List Creation (SET, SEU, MBU)
Fault Injection Campaign
Log Fault Injection Results
Fault Injection in Configuration Memory
Synthesize, Place & Route and Bitstream Generation
Configuration Frames Extraction
Essential Bits' LFAs Generation
Fault Injection using SEM IP

FIGURE 2. Complete workflow showcasing data preprocessing, model development/training/testing, HLS code generation, and fault injection in RTL and configuration memory.

A. DATA PREPROCESSING

Data preprocessing is one of the crucial initial steps before the training of DNN begins, as this step helps improve the data quality by cleaning, transforming, and normalizing it (see Fig. 2). As the problem gets more complex in MMMT CNN models, the importance of data preprocessing becomes more prevalent. CNNs perform exceptionally well on images and patterns. Therefore, the datasets need to be transformed before training. In this way, CNN can learn the common representations across different tasks from different modalities. Task 2 (T2) contains RGB image samples from the SVHN dataset, which does not require additional transformation. However, task 3 (T3) is a grayscale dataset, and task 1 (T1) is from a different modality; thus, T1 and T3 need to be preprocessed and transformed before CNN training. Data preprocessing of T1 consists of collecting the real raw radar data, extracting information by fast Fourier transform (FFT), removing the clutter, and fusing the range, velocity, and angle feature maps. Additional technical details of the real radar dataset are available in [46].

The T3 dataset is an adaptation of the MNIST dataset. The MNIST dataset in its original form contains grayscale images presented in a 2-dimensional format. The T3 dataset has been transformed into a 3-dimensional RGB image, with the added inclusion of noise to further increase the complexity of the dataset. In order to facilitate the classification of subtasks, the dataset has been expanded to include two additional colors, i.e., red and green. Finally, all the tasks are resized to the same dimensions, i.e., 32×32×3, so that there is a match between the input dimensions of the images and CNN layers.

The dataset's size and diversity can affect the model's performance. Therefore, it is important to carefully curate

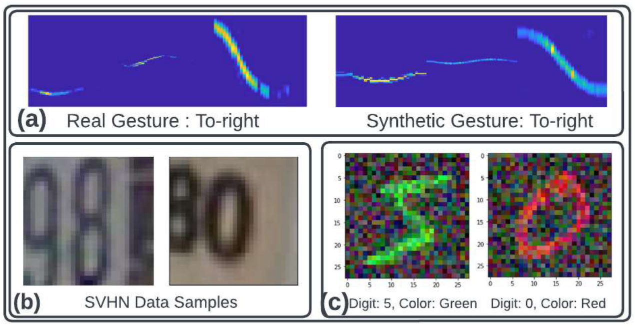

(a) Real Gesture : To-right
Synthetic Gesture: To-right
(b) SVHN Data Samples
(c) Digit: 5, Color: Green
Digit: 0, Color: Red

FIGURE 3. (a) FMCW radar hand gesture samples, {Left}: Real dataset, {Right}: Synthetic dataset (b) SVHN samples (c) Transformed MNIST dataset.

the dataset to ensure that it represents the range of variation the model will likely encounter in real-world applications. T1 consists of 1500 real radar data samples for hand gesture detection, which is way less as compared to the 99,289 T2 samples and 70,000 T3 samples. Thus, in the initial experiments, the T1 accuracy was compromised due to an imbalance between T1 and T2/T3 datasets. The real raw data samples were collected in an indoor environment with a Texas Instruments (TI) millimeter-wave radar AWR1642 and data acquisition board DCA1000. Therefore, there is a significant effort to create a real dataset of thousands of samples. In order to solve this challenge, a synthetic dataset generation approach was adopted. Synthetic data generation is the process that involves generating new data that has similar statistical properties as the original data (see Fig. 3(a)). Hence, synthetic hand gestures can be generated by utilizing Blender animation to create hand gesture trajectories and configuring Matlab's phased array system toolbox to align with AWR1642 radar parameters (refer to Fig. 4). In this way, we have increased the T1 dataset samples to 6,480 (additional technical details can be found in [45]).

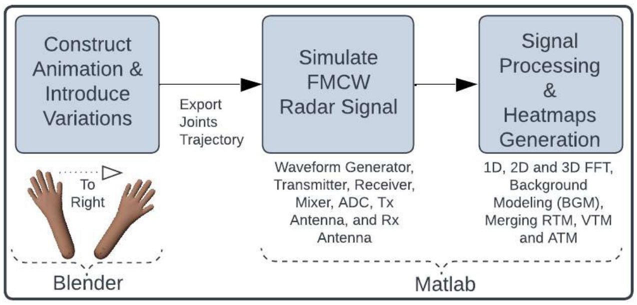

Construct Animation & Introduce Variations
Export Joints Trajectory
Simulate FMCW Radar Signal
Signal Processing & Heatmaps Generation
To Right
Waveform Generator, Transmitter, Receiver, Mixer, ADC, Tx Antenna, and Rx Antenna
1D, 2D and 3D FFT, Background Modeling (BGM), Merging RTM, VTM and ATM
Blender
Matlab

FIGURE 4. Synthetic radar dataset generation approach using Blender and Matlab.

B. MODEL CREATION, TRAINING, AND TESTING FUSED AND BRANCHED MODELS

The CNN architecture depicted in Table 3 has been created using Tensor flow/Keras and Qkeras [21]. Upon data preprocessing and model creation, training and testing of the model

TABLE 3. CNN architecture with trainable params: 14926.

Layer Name	Output Shape	Layer Name	Output Shape	Layer Name	Output Shape	Layer Name	Output Shape	Layer Name	Output Shape	Layer Name	Output Shape
InputLayer Conv2D BatchNorm. ReLU Act. MaxPooling2D	(32, 32, 3) (30, 30, 16) (30, 30, 16) (30, 30, 16) (15, 15, 16)	Conv2D BatchNorm. ReLU Act. MaxPooling2D	(13, 13, 16) (13, 13, 16) (13, 13, 16) (6, 6, 16)	Conv2D BatchNorm. ReLU Act. MaxPooling2D	(4, 4, 24) (4, 4, 24) (4, 4, 24) (2, 2, 24)	Flatten Dense BatchNorm. ReLU Act.	(,96) (,42) (,42) (,42)	Dense BatchNorm. ReLU Act.	(,64) (,64) (,64)	Dense Softmax Act.	(,26) (,26)

can be accomplished (see Fig. 2). FM training was treated as a single-task learning problem and followed a standard training process. The dataset consisted of three different tasks (i.e., T1, T2, T3), and the output layer had classes representing all three tasks. In contrast, BM training could be accomplished using the Multi-Task Learning (MTL) or Transfer Learning (TL) methods. MTL involves optimizing multiple loss functions to train multiple tasks jointly, with common layers learning shared representations between related tasks while task-specific branches perform well on specific tasks.

As all the tasks are being optimized simultaneously, it may happen that the training of all the tasks does not converge to the desired accuracy. Therefore, as an alternative, the transfer learning TL method was applied, which involved a multi-stage training process: 1) First, the entire model is trained as an FM. The last layer of the model is removed, leaving behind the layers trained on the shared representation of all three tasks. 2) Freeze the weights of the shared layers 3) Add task-specific layers to the model and train each task individually 4) if a task has additional sub-tasks, they can also be trained in a similar fashion.

Furthermore, model compression methods can also be added to the training loop. Model compression aims to reduce the model size so it can be deployed on low-power and resource-constraint edge devices without significant accuracy degradation. This is why the research community has actively pursued the topic of model compression over the last few years. Several model compression methods have been proposed in recent years, i.e., quantization, parameter pruning, low-rank factorization, knowledge distillation, transferred/compact convolutional filters, etc. Our study focuses on a) Quantization and b) Pruning, two of the most promising model compression approaches.

Through experimentation, it has been demonstrated that many of the parameters in deep neural networks are superfluous, and in fact, their absence does not significantly affect the expected performance. This situation may happen when the parameters of the neural networks are zero, close to zero, or are replicated. Thus, pruning is a way to remove unnecessary parameters that do not significantly contribute to the accuracy of results, thereby resulting in sparse deep neural networks (Fig. 7). This sparsity offers two advantages: a) it causes a significant reduction in hardware resource utilization, which further helps reduce the computational complexity, and b) It improves the resiliency of the DNN model [22]. Our implementation focuses on the magnitude-based weight pruning method to optimize the model [47].

In addition to pruning, DNN quantization is added to the training loop as it can significantly reduce the model size and increase reliability. In many cases, the dynamic range of the floating-point (FP) numbers is not needed, thus DNN quantization refers to a method of approximating a neural network's parameters and activations to low bit-width fixed point (FxP) numbers, which are hardware-friendly, offer faster computations and cost less area overhead as compared to FP computations. Additionally, quantization can enhance the DNN model's reliability, as demonstrated by [23] and [24]. Hence, we can expect significant benefits in terms of model size and reliability after DNN quantization. Both pruning and quantization lead to reduced hardware resources and power consumption and have become the de-facto step during the DNN deployment on edge devices [25]. The shared layers approach, aided by pruning and quantization, substantially reduces hardware and power resources. Table 4 and Fig. 6 illustrate the experimental results of several implementations of CNN models, while section VI delves into a comprehensive analysis of these findings.

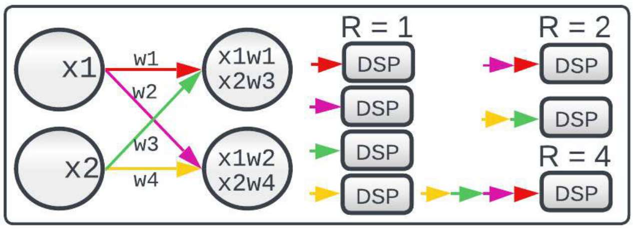


FIGURE 5. Impact of different reuse factors values on DSP48E utilization for the computation between 2 neuron pairs [26].

C. HLS4ML FRAMEWORK

The hls4ml [26] is an open-source framework that allows deploying machine learning models on FPGAs, specifically designed for low-latency and energy-efficient inference at the edge. Once a desired test accuracy is achieved, hls4ml can be used to convert the trained model into an HLS-compatible C/C++ code (see Fig. 2). The framework considers several parameters for generating the synthesizable C/C++ code. i.e., FPGA part number, interface type, reuse factor (R), and FxP precision. The parameter 'R' determines the parallelism in the hls4ml generated DNN model (see Fig. 5).

Multiplication is the most fundamental operation in neural networks as it involves multiplying the weight with the input. After the multiplication, a bias is added, and the result is passed to an activation function. A reuse factor of 1 indicates an immensely parallel design, resulting in the generation of an HLS model with the minimum achievable latency. If the reuse factor is increased by a factor N, the HLS compiler tries to reduce the DSP resource by ∼1/N, and as a result, it increases the model's overall latency by ∼N.

The parameter 'precision' is used by hls4ml to perform quantization. By default, the hls4ml employs a precision of $< 16, 6>$. This signifies that the model will be quantized to FxP 16 bits in which 6 bits are integer bits (including the sign bit) and the remaining 10 bits are allocated to fractional representation. The 'precision' parameter is modified iteratively to attain the desired level of accuracy. Hls4ml compiles the quantized model and estimates the model's accuracy (termed as hls4ml accuracy) using bit-accurate fixed point emulation of the FPGA inference code.

As FM is un-branched, it can be converted into HLS-compatible C/C++ with ease. Nonetheless, hls4ml currently lacks support for generating HLS code for BM. We recommend a workaround encompassing several sequential actions for employing this framework for BM. 1) Dissect the BM in linear branches. 2) Utilizing hls4ml, create HLS code for each branch individually. 3) Generate HDL code of all branches by utilizing Vivado HLS. 4) Stitch all branches in HDL using control element (CE) (Fig. 1(b)). CE, 1) synchronize the data streams and control signals between the stitched branches, and 2) enable/disable single or multiple branches based on the tasks.

TABLE 4. Experimental results of fused and branched models.

Model	Accuracy (%)	HLS4ML Accuracy (%)	Quantization Type	Pruning (%)	Latency (us)	Total Latency(us)
FM	T1 = 94.67 T2 = 86.33 T3 = 93.78	T1= 93.33 T2 = 86.4 T3= 93.67	PTQ (BW= 20,10)	0	T1= 5.21 T2= 5.21 T3= 5.21	15.63
FMP	T1= 100 T2= 90.53 T3= 95.46	T1= 100 T2= 90.38 T3= 95.51	PTQ (BW= 20,10)	50	T1= 5.21 T2= 5.21 T3= 5.21	15.63
FMQ	T1= 94.00 T2= 88.40 T3= 93.55	T1= 92.00 T2= 87.97 T3= 91.77	QAT (Varying BW)	0	T1= 5.21 T2= 5.21 T3= 5.21	15.63
FMQP	T1= 97.33 T2= 89.22 T3= 94.36	T1= 96.67 T2= 89.14 T3= 93.91	QAT (Varying BW)	CNN=53 Dense=75	T1= 5.21 T2= 5.21 T3= 5.21	15.63
BMQP	T1 = 98.00 T2 = 89.31 T3 = 95.33 T3c=96.24	T1 = 97.33 T2 = 89.31 T3 = 95.39 T3c= 96.22	QAT (Varying BW)	50-85(Vary for different branches)	T1= 5.205 T2= 5.270 T3= 5.210 T3c= 5.09	15.685

VI. EXPERIMENTAL RESULTS AND DISCUSSION OF FUSED AND BRANCHED MODELS

In accordance with the workflow outlined in the section V, we have devised five distinct CNN models: fused model (FM), fused model pruned (FMP), fused model quantized (FMQ), fused model quantized and pruned (FMQP), and branched model quantized and pruned (BMQP). Table 4 illustrates the experimental results of accuracy and latency, while Fig. 6 portrays the power, energy, and hardware resource utilization savings factor relative to baseline FM implementation. The savings factor (SF) quantifies how many times a model's power, energy, and hardware resources have reduced compared to the baseline FM. It is a ratio of the baseline FM value to the value of the model under consideration. Thus, the highest SF value suggests the most efficient model. The absolute FM values are also indicated in Fig. 6, providing a reference for estimating the absolute values of the other models. The latency of all models was calculated through RTL simulation, and power was estimated during post-synthesis functional simulation using a switching activity file of the design (i.e., SAIF file). All five models are synthesized in Vivado, targeting Xilinx Virtex UltraScale+ VCU118 at 200 MHz clock frequency. Notably, all models have demonstrated over 90% average accuracy across all three tasks on a single application-specific accelerator, which distinguishes our work from prior studies where only one accelerator is optimized to perform just one task.

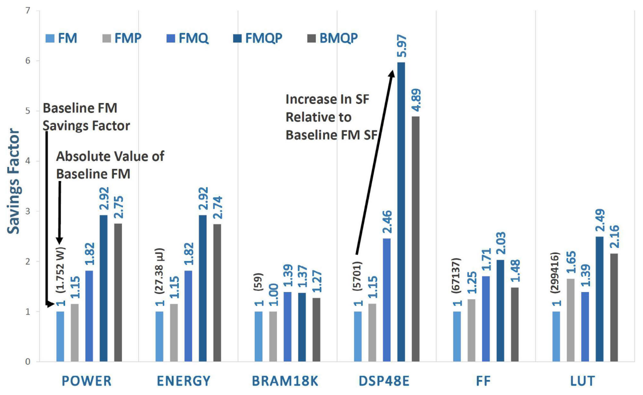


FIGURE 6. Power, energy, and hardware resource utilization savings factor of FMP, FMQ, FMQP, and BMQP relative to baseline FM.

The non-optimized baseline implementation is FM, which has good accuracy for all three tasks but utilizes the most hardware resources, power, and energy (i.e., SF = 1). FMP is a 50% pruned version of the baseline FM. The high accuracy of FM and FMP comes at the cost of increased resource utilization, power, and energy consumption as these two models are post-training homogeneously quantized to $< 20, 10>$ bit-widths (BW). In PTQ, the model is first trained using a floating-point 32-bit (FP32) data type, and then the model's weights and activation functions are quantized. This method is simple to apply, but it could decrease accuracy due to a loss of dynamic range. This is why PTQ FM and FMP require a higher bit-width (BW). This provides both FM and FMP with a good dynamic range and helps avoid any saturation and wrap-around issues.

FMQ is trained directly with lower precision (i.e., quantization-aware training (QAT)), and any loss in accuracy due to the loss of precision can be rectified during the training process, enabling the model to perform better in low BW. FMQ performs equally well in terms of accuracy while consuming fewer hardware resources, power, and energy. DSPs are a scarce resource, and these DSP hard macros consume a higher amount of power compared to LUTs. While synthesizing FMQ trained on BW using QAT, the simpler multiplication operations are mapped on LUTs instead of DSPs. This reduces DSP utilization, power consumption, and overall energy usage, as also evident from improved FMQ SF values compared to FM and FMP.

FMQ and FMP have different strengths. FMP removes the unnecessary weights, but the existing weights have high BW due to homogeneous PTQ. FMQ has low BW due to heterogeneous QAT, but this model is not pruned. FMQP (see Fig. 7) combines the strength of heterogeneous QAT and pruning to achieve the best possible results in terms of accuracy, hardware resources, power, and energy consumption. (i.e., highest SF value). BMQP is obtained using the multi-stage transfer learning training method, and its HLS code is generated using the workaround explained in the previous section. BMQP delivers slightly higher accuracy as compared to FMQP. In BMQP, multiple branches can be active simultaneously, thereby supporting the classification of numerous sub-tasks. Thus, to illustrate the multi-task support in BMQP for task 3, two branches get active to classify digits (T3) and the color of the digits (T3c).

In comparison with FMQP, BMQP SF is lower, as it consumes slightly more hardware resources, power, and energy, with additional benefits in the form of resilience and multi-task support. The impact of quantization and pruning is the same in BM; therefore, only the results of BMQP are shown. Pruning helps reduce hardware resources and improves the model's accuracy, as evidenced by the FMP, FMQP, and BMQP results.

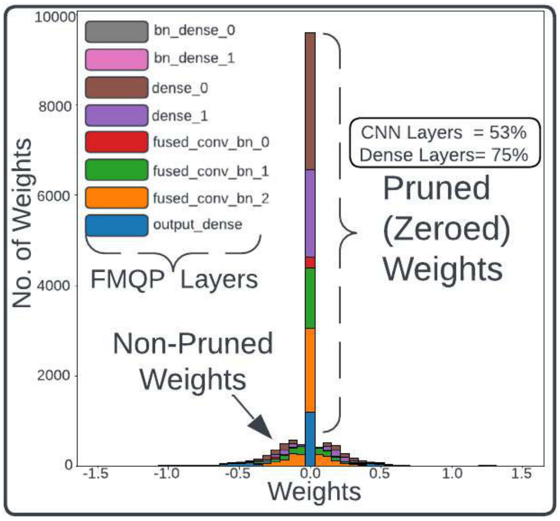


FIGURE 7. FMQP model weights distribution.

All the fused models have similar latency. The latency of the hls4ml generated CNN architecture depends on the network's depth, reuse factor ($R = 1$) of DSPs, and input size ($32\times32\times3$). All three parameters are similar for FM, FMP, FMQ, and FMQP. The execution of the layers is sequential, which means that the subsequent layers can only process the data when the previous layer finishes its computation. Therefore, for hls4ml, it is advisable to have a wider network (more kernels/parameters in a layer) as opposed to more layers (depth of the DNN), as it is more efficient to parallelize per-layer computations in FPGA. Latency is not impacted by the quantization bit widths, kernel size, and the number of

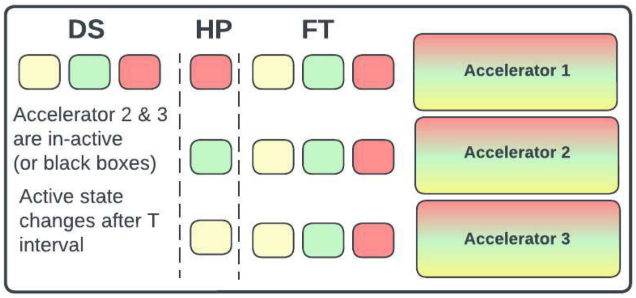
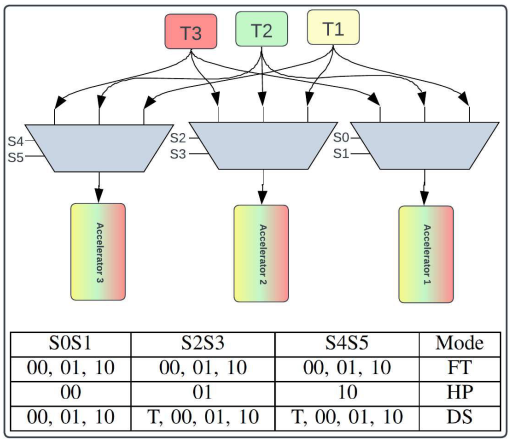

DS
HP
FT
Accelerator 2 & 3 are in-active (or black boxes)
Active state changes after T interval
Accelerator 1
Accelerator 2
Accelerator 3

FIGURE 8. Reconfigurable CNN accelerators.

kernels. This is why fused models being different in terms of pruning percentage and quantization bits, have similar latency. The latency of each task varies in BMQP, as each task goes through only specific layers. i.e., T3c branched earliest and has the lowest latency, while T2 branched at the last layer and therefore has the highest latency in BMQP. T3c latency is not added to the total latency, as T3 and T3c branches execute in parallel. Our hardware results match the analysis presented in [26]. Achieving the best match of hardware resource utilization, accuracy, power, energy, and latency depend on the specific application and the available hardware resources in the targeted hardware.



S0S1	S2S3	S4S5	Mode
00, 01, 10	00, 01, 10	00, 01, 10	FT
00	01	10	HP
00, 01, 10	T, 00, 01, 10	T, 00, 01, 10	DS

FIGURE 9. Switching mechanism between different operating modes.

VII. RECONFIGURABLE MULTI-MODAL CNN ACCELERATOR

Reconfigurability is essential in a dynamic world, as it enables flexibility, agility, and customization to meet ever-changing AI computation needs and requirements. Our perspective on reconfigurability is based on hardware reliability and computing performance. Execution of multiple tasks in different modes is often associated with microprocessors [27], [28]. The proposed approach of shared layers makes it possible to execute multiple tasks on an application-specific CNN accelerator during runtime. The hls4ml accelerator does not store weights in a separate memory location. In this hls4ml accelerator generation framework, the neural network

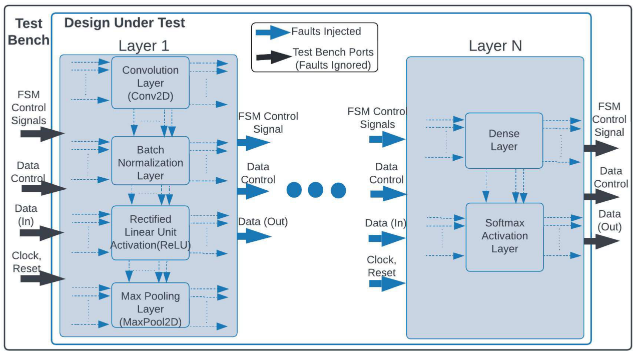


FIGURE 10. RTL fault injection at the ports level.

TABLE 5. SET fault injection analysis for diverse pulse width.

Frequency (MHz)	SET PW (ps)	Faults Injected	Faults Detected	SETFIR (%)
200	100	3000	0	0
200	250	3000	0	0
200	500	3000	31	1.03
200	750	3000	31	1.03
200	1000	3000	31	1.03

accelerator weights are fused in the network architecture. This eliminates the necessity to fetch weights from a remote memory location. A single (or multiple) instance of an accelerator can process any task (tasks on which it is trained) in any operating mode because the value of weights results from a training process that includes all three tasks.

By default, one accelerator is sufficient to perform three tasks, but with triple modular redundancy (TMR), incorporated with a simple mux-based design (Fig. 9), it is possible to execute tasks in 1) Fault tolerant (FT), 2) High performance (HP), and 3) De-stress(DS) modes (Fig. 8). Among the five models illustrated in Table 4, FMQP and BMQP are the most optimized fused and branched models. As, for FT, HP, and DS operating modes, redundant instances of FMQP/BMQP are utilized, the accuracy of FMQP/BMQP models in all operating modes will remain the same (as illustrated in Table 4) in all operational modes.

A. FT MODE

In FT mode, all tasks are executed on all three accelerators, providing high reliability (Fig. 8 and 9). The latency, power, and energy in FT mode have been depicted in Table 7. The hardware resource utilization in the case of DMR and TMR can be estimated using the values already mentioned in Fig. 6, as it will be a multiple two in DMR and three in the case of TMR. Among the five models, FMQP and BMQP are the most optimized fused and branched models, respectively. We have performed a comprehensive fault analysis of FMQP and BMQP at the RTL level and in the configuration memory of the FPGA.

1) FAULT INJECTION AT RTL LEVEL

The HDL code generated by Vivado HLS is imported into Vivado. Since Vivado is compatible with third-party simulators, the Xcelium fault simulator is utilized for conducting a thorough fault injection at the RTL level (see Fig. 2). There are approximately 1.2 million nodes where the fault can be injected. Injecting one fault takes around 6 seconds, and for 1.2 million nodes, it will take approximately 83 days. The overall simulation time can drastically increase if we take into account multiple models (i.e., FMQP, BMQP) operating in different modes (i.e., DMR, TMR) to study the influence of Single Event Transients (SETs), Single Event Upsets (SEUs), and Multi-bit upsets (MBUs) fault models. Thus, instead of targeting every fault node of all the components, we have targeted the ports of RTL modules (see Fig. 10). In this manner, the total number of fault nodes reduces to 35912 for FMQP and 36710 for BMQP, which takes approximately 2.5 days.

We have considered three fault models, i.e., SETs, SEUs, and MBUs. The SET-induced failure rate (SETFIR) is defined as the percentage of SET faults propagating to the output, thereby leading to an incorrect result. Similarly, SEUFIR and MBUFIR are estimated. The metrics SETFIR, SEUFIR, and MBUFIR can assist in determining the resilience of the model.

The phenomenon of the SETs occurs when a high-energy particle strikes a combinational circuit and causes a transient voltage disturbance due to charge deposition. If the particle's energy crosses a certain threshold, the end effect of it is a SET (Glitch) in the combination circuit. If the same SET propagates to a storage element and gets latched, it becomes an SEU. When a single SEU affects multiple bits, it is considered a Multi-bit upset (MBU). Thus to estimate the failure rate due to MBUs (MBUFIR), in the Xcelium fault simulator MBU can be emulated as 2 SEUs injected simultaneously.

Pulse width (PW) duration matters in SET fault analysis. SETs of shorter pulse widths may not be harmful. Therefore, before performing a comprehensive SET analysis on all ports, we first investigated the FMQP on various pulse width duration on a subset of randomly selected ports (see Table 5). SETs of Pulse widths of 100ps and 250ps did not lead to incorrect output (i.e., 0% SETFIR). Out of all the faults injected, approximately 1% of SETs consisting of PW 500ps, 750ps, and 1000ps, lead to incorrect results at the output. 1/3rd of this 1% SETFIR consists of reset ports. This implies that reset lines are very critical even at 1% SETFIR. Typically, the longer the PW, the greater the probability that SET will get latched at the clock edge. The studies, [29], [30], [31], indicate that Propagation Induced Pulse Broadening (PIPB) may result in significant SET broadening, and the resulting SETs may be longer than 1000ps. Thus, conservatively, we have selected PW 1000ps for further extensive analysis of both FMQP and BMQP. Table 6 illustrates the fault injection results of both architectures under the influence of SETs, SEUs, and MBUs. The faults are injected during the runtime of all three tasks. The salient points of the fault analysis are

- Under the influence of all the faults models (SETs, SEUs, MBUs), BMQP is slightly more resilient (roughly

TABLE 6. Fault injection analysis of FMQP and BMQP.

Model	Faults Injected	SEUs Detected	SEUFIR (%)	SETs(PW:1000ps) Detected	SETFIR (%)	MBUs Detected	MBUFIR (%)	SEUs Resets (%)	SETs Resets (%)	MBUs Resets (%)	DMR	TMR
FMQP	35912	6021	16.76	351	0.977	10896	30.34	9.5	30.19	5.19	Detected	Corrected
BMQP	36710	5383	14.6	293	0.79	9821	26.75	9.47	29.01	5.29	Detected	Corrected

2-3.6 %) than FMQP because of its branched architecture.

- 0.97 % of SETs (PW:1000ps) lead to incorrect results in FMQP, while this percentage is 0.79% in BMQP. For both models, of all the ports where SETs lead to incorrect output, approximately 30% of them are reset ports (i.e., SETs Resets (%)). This suggests the criticality of the reset lines even in low SETFIR. This percentage is approximately 9.5 % in SEUs and 5.2 % in MBUs for both models.
- MBUFIR is higher as compared to SEUFIR and SETFIR (PW:1000ps). MBUFIR is the highest among the three fault models because of the high fault rate (i.e., two SEUs). SETFIR is the lowest, as the chances of fault masking in SETs are generally higher. The SET fault propagation depends upon the SET PW (see Table 5) and the frequency of the design. At higher frequencies, there will be more clock edges, which means more chances for SETs to be captured by flip-flops [29].
- FT mode can operate in DMR (i.e., fail-safe) and TMR (i.e., fail-operational) mode. The DMR mode can detect all the faults using a comparator circuit. In comparison, TMR uses a majority voter TMR mode to detect and produce correct results at the output. Using traditional approaches to achieve FT mode (TMR) for three tasks will require nine CNN accelerators.
- The total latency (TL) is 15.63 us, translating to 326.6 GOPs[1]. The detailed experimental results of latency, power, and energy consumption in the FT mode are illustrated in Table 7.

TABLE 7. Experimental results of HP, FT, and DS modes.

Mode	Model	Latency (us)	Power (W)	Energy (uj)	GOPs/TOPs
HP	FMQP	T1 = 5.21 T2 = 5.21 T3 = 5.21 TL = 5.21	Accel 1 = 0.551 Accel 2 = 0.579 Accel 3 = 0.593	Accel 1 = 2.870 Accel 2 = 3.016 Accel 3 = 3.089	0.979 TOPs
	BMQP	T1 = 5.205 T2 = 5.270 T3 = 5.210 TL= 5.270	Accel 1 = 0.588 Accel 2 = 0.618 Accel 3 = 0.66	Accel 1 = 3.098 Accel 2 = 3.256 Accel 3 = 3.478	
FT	FMQP	T1 = 5.21 T2 = 5.21 T3 = 5.21 TL = 15.63	Accel 1 = 0.599 Accel 2 = 0.599 Accel 3 = 0.599	Accel 1 = 9.362 Accel 2 = 9.362 Accel 3 = 9.362	326.6 GOPs
	BMQP	T1 = 5.205 T2 = 5.270 T3 = 5.210 TL =15.685	Accel 1 = 0.637 Accel 2 = 0.637 Accel 3 = 0.637	Accel 1 = 9.991 Accel 2 = 9.991 Accel 3 = 9.991	
DS	FMQP	T1 = 5.21 T2 = 5.21 T3 = 5.21 TL =15.63	Accel 1 = 0.599 Accel 2 = 0.393 (CG), 0 (PR) Accel 3 = 0.393 (CG), 0 (PR)	Accel 1 = 9.362 Accel 2 = 6.142 (CG), 0 (PR) Accel 3 = 6.142 (CG), 0 (PR)	108.88 GOPs
	BMQP	T1 = 5.205 T2 = 5.270 T3 = 5.210 TL =15.685	Accel 1 = 0.637 Accel 2 = 0.437 (CG), 0 (PR) Accel 3 = 0.437 (CG), 0 (PR)	Accel 1 = 9.991 Accel 2 = 6.854 (CG), 0 (PR) Accel 3 = 6.854 (CG), 0 (PR)	

[1]GOPs = Giga Operations Per Second, TOPs = Tera Operations Per Second

Ultimately, Deep Neural Networks (DNNs) will be implemented on hardware. Therefore, analyzing the neural networks under the influence of hardware faults is essential. Numerous factors can affect the resiliency of deep neural networks. i.e., quantization, layer type, network architecture, the bit position of weights, pruning, data type, etc. For instance, consider a weight parameter value of a DNN model represented as a floating-point 32 (FP32) number. A fault in the most significant exponent bit of the FP32 number can substantially change the value of the DNN's parameter and dramatically decrease the accuracy [36]. The majority of studies have only considered fault injection in the weights of the neural network. Reference [12] argues that the reliability of a DNN model is also influenced by the hardware architecture implemented on the targeted hardware (i.e., ASICs or FPGAs). For instance, faults in the shared buffers may be read multiple times because of the reuse (output feature maps, reuse of weights, input feature maps, etc.), and therefore the same erroneous value can be distributed to multiple locations very quickly [12]. Therefore, for safety-critical applications, it becomes paramount to do an exhaustive design analysis. The fault analysis results of this study on SET, SEU, and MBU fault models provide a good insight into the resiliency of the CNNs hardware accelerator architecture.

2) FAULT INJECTION IN CONFIGURATION MEMORY

In ASICs, the logic is permanently mapped to gates and flip-flops in silicon. Whereas in FPGAs, logic is mapped on the configurable logic blocks (CLBs), which consist of Lookup Tables (LUTs), flip-flops (FFs), and routing resources (switch matrix, multiplexors, etc.). Unlike ASICs, FPGAs offer programmability, enabling alteration of functionality by loading a new bitstream. The bitstream comprises configuration frames that set up all programmable and memory elements within the FPGA fabric. However, it's important to recognize that these bits are susceptible to radiation-induced effects, necessitating thorough scrutiny against diverse fault scenarios.

Authors in [48] have investigated that the impact of the fault in FPGAs is sometimes different as in ASICs and for the reason that FPGA fabric is a bit different compared to ASICs. I.e., The end effect of SEUs in the configuration memory (CRAM) of FPGAs, and it can be mitigated by bitstream reconfiguration. To evaluate the impact of SEU on the CRAM in FT mode, a fault injection platform has been built based on ZCU102. Furthermore, this platform has the ability to conduct dynamic partial reconfiguration of the FPGA bitstream, intended for utilization in DS mode.

CRAM of the FPGA consists of thousands of frames consisting of essential and non-essential bits. FI is performed on the essential bits, which are considered critical for the system's functionality, employing the subsequent procedure: 1) Post bitstream generation (see Fig. 2), Configuration frame numbers corresponding to the accelerator's instance are extracted from the logic location file. 2) Linear Frame addresses (LFAs) are generated using frame numbers and essential bits of information. 3) Soft Error Mitigation (SEM) IP core [49], controlled from the application software running on the Zynq processor, performs fault injection in the CRAM. Apart from facilitating fault injection, the SEM IP core can also carry out fault detection and correction. The primary focal points regarding fault injection in the CRAM are outlined as follows:

- FI is performed on randomly selected 4000 different essential bits LFAs. SEM IP continuously monitors the CRAM and repairs the faulty bit using ECC and CRC-based algorithms. Depending upon the fault location, fault detection/correction latency using SEM IP core varies between 3.14 ms to 48.96 ms. This latency further depends on the clock frequency, FPGA size [49].
- The SEM IP is designed to detect and correct SEUs that occur within the CRAM. Notably, the SEM IP core does not capture SETs occurring in the combinational logic nor identifies SEUs/MBUs in the sequential logic. Thus, SEM IP combined with the TMR/DMR-based majority voter design can cover various fault models, i.e., SETs, SEUs, MBUs, and SEUs in CRAM.

B. HP MODE

This mode executes all the tasks in parallel, i.e., each accelerator executes the individual task (Fig. 8 and 9), which reduces total latency (TL) by 2x. The experimental results during the HP mode are demonstrated in Table 7. The major takeaway from HP mode is

- The latency is reduced by 2x, i.e., 15.63 us to 5.21us in FMQP and from 15.685us to 5.27us in BMQP (considering all the BMQP accelerators get input at the same time), translating to having 0.979 TOPs compute power.
- Each accelerator's power and energy consumption slightly varies, as in HP mode, each accelerator is processing different input samples. Compared to FT mode, the reduction in energy consumption is approximately 2x.
- Because of shorter overall latency, the combined energy consumption of all accelerators in HP mode is less than in DS mode (CG and PR).

C. DS MODE

In this aging-aware mode, only one accelerator is operational at a time, and tasks are executed sequentially using a TDM approach.

Alongside SEUs, SETs, and MBUs due to high-energy particles, transistor aging is another significant reliability concern in integrated circuits. Aging refers to the gradual degradation (i.e., transistor's threshold voltage and a decrease in the carrier mobility) of circuit performance over time due to Hot Carrier Injection (HCI) and Bias Temperature Instability (BTI) [32], [33]. This results in a slowdown in the transistor's switching speed, which further leads to a high number of timing failures, thereby reducing the overall operational lifetime of the circuit. The aging rate is influenced by multiple factors, i.e., supply voltage, switching activity, temperature, transistor stress duration, etc. In this study, we have redundant instances of accelerators, which are helpful in scenarios where high computation (HP mode) and high reliability (FT mode) are required. If these modes are inactive, the redundant accelerators experience persistent stress, which can contribute to aging. Every time the clock signal transitions, the elements of the circuit (i.e. flip-flops, combinational logic gates, etc.) may also change their state, which results in power consumption. The power consumption due to the switching activity of the circuit is described as dynamic power. In such a scenario, a power optimization method, clock gating (CG), can disable the clock signal to idle accelerators to reduce unnecessary switching activity. This diminishes the dynamic power consumption, which translates to decelerating the aging process. Additionally, we have investigated partial reconfiguration (PR) methodology to reduce the circuit elements' stress. The main aim of the PR method is to keep only the accelerator configured and erase the configuration area of the idle accelerators. This can be achieved using the dynamic partial bitstream reconfiguration workflow in AMD Xilinx FPGAs. The brief workflow consists of 1) Assign a reconfigurable region (RR) for each accelerator on the FPGA fabric, 2) Generate partial bit streams of all RRs, 3) While one accelerator in one RR is active, the other two RRs contain nothing (i.e., black boxes). In DS mode, the total latency is 15.63 us, equivalent to 108.88 GOPs. The detailed experimental results are illustrated in Table 7, and the essential takeaway of comparison between the CG and PR for DS mode is as follows:

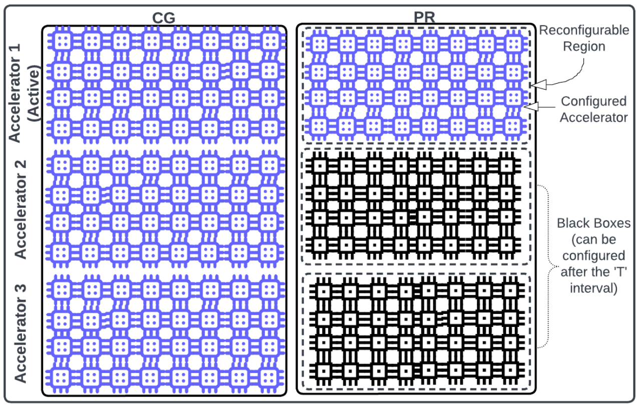


FIGURE 11. FPGA fabric {Left}: in a CG design {Right}: in a PR design.

- There is approximately 65.6% and 68.6 % decrease in the dynamic power (CG) consumption in idle

TABLE 8. Quantitative comparison with prior ASICs and FPGA-based implementations.

	FPGAs							ASICs				
	Proposed	[10]	[8]	[18]	[17]	[26]	[3]	[4]	[37]	[7]	[9]	[5]
Target Hardware	VCU118	ZC706	ZYNQ7045	ZCU102	ZU9	XC7Z020	ZU3EG Ultra96	ASIC 90nm	ASIC 28nm	ASIC 65nm	ASIC 65nm	ASIC 65nm
Model	6 Layer CNN	9 layer CNN	VGG16	DeepSense	ResNet-18	6 layer CNN	CNVW2A2	Tiny-Yolov2	VGG16	Autoencoder	AlexNet	LeNet-5
Accuracy(%)	T1= 97.33 T2= 89.22 T3= 94.36	80.1	67.60%	94.19	-	94	-	-	-	-	-	-
Task(s)	3 (SVHN, MNIST, Radar)	1 (CIFAR-10)	1 (ImageNet)	1 (HHAR)	3 (Cityscapes)	1 (SVHN)	1 (CIFAR-10)	1	1	1	1	1 (Traffic Signs)
Sensor Modalities	2	1	1	2	1	1	1	1	1	1	1	1
Performance (Latency, GOPs, TOPs, images/s)	15.63 us - 5.21 us 108.88 GOPs - 0.979 TOPs 63979.5 - 575815.7 images/s	283 us, 21906 images/s	81 images/s	10 ms	13.6 GOPs, 55 images/s	171 µs, 2831 images/s	4863 us, 205.64 images/s	14.4 GOPs	10 TOPs	50 ns	194.4 GOPs	152 GOPs
Partial Reconfiguration	Yes	-	-	-	-	-	Yes	-	-	-	-	-
LUTs	120202	46253	186280	-	-	48259	36860	-	-	-	-	-
FFs	33015	-	-	-	-	35118	46217	-	-	-	-	-
DSPs	955	-	796	-	-	213	32	-	-	-	-	-
BRAMs	43	183	362	-	-	122	154.5	-	-	-	-	-
SRAM	-	-	-	-	-	-	-	322KB	144KB	-	280KB	96 KB
Gate Count/Area	-	-	-	-	-	-	-	140K	1.95 M	800k	16 mm2	1.3 M
Power (W)	0.599	3.6	-	-	12.1	-	1.876	-	0.3	0.095	0.479	0.35
Energy ((uJ or mJ)/sample))	9.362 uJ	-	-	50 mJ	-	-	9.123 mJ	-	-	2.4 nJ	-	-
Frequency (MHz)	200	200	240	600	-	100	250	100	200	40	200	500
Model Compression	Pruning, Quantization	Quantization	Quantization	Quantization	-	Pruning, Quantization	Quantization	Quantization	Quantization	Quantization	Quantization	Quantization

FMQP and BMQP accelerator, respectively. This aids in decelerating the aging process.

- The components such as DSPs, Block RAMs (BRAMs), and Logic are the biggest consumers of dynamic power, accounting for around 86% of total power consumption. DSP units are the most power-hungry among these components, followed by BRAM. Logic, which includes LUTs and registers, ranks third.(in all the designs).
- The FPGA device static power represents the transistor leakage power and is a function of process, voltage, and temperature [34]. This is approximately the same for all the designs, i.e., ∼2.5W.
- Using the CG method, the percentage of dynamic power reduction in ASICs is more as compared to the FPGAs, for the reason that, in ASICs, there is more freedom and control over clock trees design [35].
- Our experimental results display that a single idle accelerator still consumes 34.5 % of dynamic power, causing aging in the circuit. Removing the idle accelerators using PR can ensure maximum power savings in such a scenario. After a specific time interval (T), an existing accelerator is erased, and the next-in-line accelerator is dynamically configured on the assigned RR (see Fig. 11). Hence, the PR method can decelerate aging more as compared to the standard CG method.
- It takes approximately 34 ms to perform PR of a single accelerator instance using the processor configuration access port (PCAP) interface in the fault injection platform. This latency varies depending on the size of the RR, clock frequency, and size of the FPGA and interface used for PR (ICAP/PCAP). In this case, The PR is scheduled for execution during system inactivity or standby mode periods, ensuring that latency-related safety risks are effectively mitigated.

D. COMPARISON WITH PRIOR STUDIES

Table 8 provides a quantitative comparison of our FPGA-based implementation results with the previous studies. The comparison consists of neural network deployment for both FPGAs and ASICs-based implementations. For simplicity in comparison, we have utilized the FMQP model for comparison. Our methodology extracts more performance from the CNN accelerator than other prior studies. The performance of the accelerator varies depending on the operating mode. DS, FT, and HP modes offer 108.88 GOPs, 326.6 GOPs, and 0.979 TOPs, respectively.

For FPGA-based designs, our approach can process multiple tasks power-efficiently while maintaining a good performance even in the DS mode. E.g. In comparison with [26], which uses almost the same CNN model as ours, it is processing only one task from one modality. In contrast, our approach can process 3 tasks from 2 different sensor modalities. As compared to other implementations based on FPGAs, our findings demonstrate competitiveness in terms of performance and power/energy, as well as the added advantages of reconfigurability/self-adaptivity.

Generally, for a specific application, ASIC implementations are more power/energy efficient than implementing the same design on FPGA. However, apart from [7] and [37], our findings remain competitive in comparison to other ASIC implementations. Additionally, our approach provides enhanced support for re-reconfigurability, addressing the dynamic requirements of AI applications in both high-performance and high-reliability domains, as noticeable from the comprehensive comparison in Tables 1, 2 and 8.

VIII. CONCLUSION AND FUTURE WORK

The shared-layers methodology utilizes the fundamental operating principles of CNNs to execute three distinct tasks on a single optimized multi-modal multi-task application-specific CNN accelerator. By incorporating redundancy, this approach is further leveraged to propose a fault-tolerant accelerator with reconfigurable capabilities suitable for operating in FT, DS, and HP modes. The detailed workflow, methodology, experimental results, and fault injection analysis are presented, demonstrating our approach's effectiveness and versatility compared to the state-of-the-art. This work serves as an essential step towards developing more efficient, reconfigurable, and intelligent AI processing systems that adapt to changing AI application requirements.

Future work will focus on achieving a fully reconfigurable AI processing system consisting of four major building blocks: 1) On-chip sensors (temperature, aging, and SEU sensors, etc.), 2) Reconfigurable quad-core RISC-V cores, 3) Reconfigurable AI accelerators, and 4) Reconfigurable hardware (i.e., FPGAs). These building blocks will assist in dynamically reconfiguring the RISC-V processing cores, AI accelerators, and RRs on the FPGA fabric to optimize performance, energy reduction, and reliability improvement.

REFERENCES

[1] R. T. Syed, M. Andjelkovic, M. Ulbricht, and M. Krstic, "Towards reconfigurable CNN accelerator for FPGA implementation," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 70, no. 3, pp. 1249–1253, Mar. 2023, doi: 10.1109/TCSII.2023.3241154.

[2] C. H. Yi, K. Kwon, and J. W. Jeon, "Method of improved hardware redundancy for automotive system," in *Proc. 14th Int. Symp. Commun. Inf. Technol. (ISCIT)*, Sep. 2014, pp. 204–207.

[3] O. Eldash, A. Frost, K. Khalil, A. Kumar, and M. Bayoumi, "Dynamically reconfigurable deep learning for efficient video processing in smart IoT systems," in *Proc. IEEE 6th World Forum Internet Things*, New Orleans, LA, USA, Jun. 2020, pp. 1–6, doi: 10.1109/WF-IoT48130.2020.9221101.

[4] C.-B. Wu, C.-S. Wang, and Y.-K. Hsiao, "Reconfigurable hardware architecture design and implementation for AI deep learning accelerator," in *Proc. IEEE 9th Global Conf. Consum. Electron. (GCCE)*, Kobe, Japan, Oct. 2020, pp. 154–155, doi: 10.1109/GCCE50665.2020.9291854.

[5] L. Du, Y. Du, Y. Li, J. Su, Y.-C. Kuan, C.-C. Liu, and M. F. Chang, "A reconfigurable streaming deep convolutional neural network accelerator for Internet of Things," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 1, pp. 198–208, Jan. 2018, doi: 10.1109/TCSI.2017.2735490.

[6] J. Cho, Y. Jung, S. Lee, and Y. Jung, "Reconfigurable binary neural network accelerator with adaptive parallelism scheme," *Electronics*, vol. 10, no. 3, p. 230, Jan. 2021.

[7] G. D. Guglielmo, F. Fahim, C. Herwig, M. B. Valentin, J. Duarte, C. Gingu, P. Harris, J. Hirschauer, M. Kwok, V. Loncar, Y. Luo, L. Miranda, J. Ngadiuba, D. Noonan, S. Ogrenci-Memik, M. Pierini, S. Summers, and N. Tran, "A reconfigurable neural network ASIC for detector front-end data compression at the HL-LHC," *IEEE Trans. Nucl. Sci.*, vol. 68, no. 8, pp. 2179–2186, Aug. 2021.

[8] M. P. Véstias, R. P. Duarte, J. T. De Sousa, and H. C. Neto, "A configurable architecture for running hybrid convolutional neural networks in low-density FPGAs," *IEEE Access*, vol. 8, pp. 107229–107243, 2020, doi: 10.1109/ACCESS.2020.3000444.

[9] F. Tu, S. Yin, P. Ouyang, S. Tang, L. Liu, and S. Wei, "Deep convolutional neural network architecture with reconfigurable computation patterns," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 8, pp. 2220–2233, Aug. 2017.

[10] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "FINN: A framework for fast, scalable binarized neural network inference," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, Feb. 2017, pp. 65–74, doi: 10.1145/3020078.3021744.

[11] S. Srinivasan, P. Mangalagiri, Y. Xie, N. Vijaykrishnan, and K. Sarpatwari, "FLAW: FPGA lifetime awareness," in *Proc. 43rd Annu. Conf. Design Autom.*, Jan. 2006, pp. 630–635, doi: 10.1145/1146909.1147070.

[12] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," in *Proc. Int. Conf. for High Perform. Comput., Netw., Storage Anal.*, Nov. 2017, pp. 1–12, doi: 10.1145/3126908.3126964.

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

[14] L. K. Draghetti, F. F. D. Santos, L. Carro, and P. Rech, "Detecting errors in convolutional neural networks using inter frame spatio-temporal correlation," in *Proc. IEEE 25th Int. Symp. On-Line Test. Robust Syst. Design (IOLTS)*, Jul. 2019, pp. 310–315.

[15] J. J. Zhang, T. Gu, K. Basu, and S. Garg, "Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator," in *Proc. IEEE 36th VLSI Test Symp. (VTS)*, Apr. 2018, pp. 1–6.

[16] G. Gambardella, J. Kappauf, M. Blott, C. Doehring, M. Kumm, P. Zipf, and K. Vissers, "Efficient error-tolerant quantized neural network accelerators," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Oct. 2019, pp. 1–6.

[17] J. Peng, L. Tian, X. Jia, H. Guo, Y. Xu, D. Xie, H. Luo, Y. Shan, and Y. Wang, "Multi-task ADAS system on FPGA," in *Proc. IEEE Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Hsinchu, Taiwan, Mar. 2019, pp. 171–174, doi: 10.1109/AICAS.2019.8771615.

[18] M. T. Ajili and Y. Hara-Azumi, "Multimodal neural network acceleration on a hybrid CPU-FPGA architecture: A case study," *IEEE Access*, vol. 10, pp. 9603–9617, 2022, doi: 10.1109/ACCESS.2022.3144977.

[19] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proc. NIPS Workshop Deep Learn. Unsupervised Feature Learn.*, 2011, pp. 1–9.

[20] Y. LeCun and C. Cortes. (2010). *MNIST Handwritten Digit Database*. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[21] C. N. Coelho, A. Kuusela, S. Li, H. Zhuang, T. Aarrestad, V. Loncar, J. Ngadiuba, M. Pierini, A. A. Pol, and S. Summers, "Ultra low-latency, low-area inference accelerators using heterogeneous deep quantization with QKeras and hls4ml," 2020, *arXiv:2006.10159*.

[22] Z. Gao, X. Wei, H. Zhang, W. Li, G. Ge, Y. Wang, and P. Reviriego, "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.

[23] B. F. Goldstein, S. Srinivasan, D. Das, K. Banerjee, L. Santiago, V. C. Ferreira, A. S. Nery, S. Kundu, and F. M. G. França, "Reliability evaluation of compressed deep learning models," in *Proc. IEEE 11th Latin Amer. Symp. Circuits Syst. (LASCAS)*, Feb. 2020, pp. 1–5.

[24] M. Sabbagh, C. Gongye, Y. Fei, and Y. Wang, "Evaluating fault resiliency of compressed deep neural networks," in *Proc. IEEE Int. Conf. Embedded Softw. Syst. (ICESS)*, Jun. 2019, pp. 1–7.

[25] (2022). *Xilinx Vitis AI Overview*. [Online]. Available: https://docs.xilinx.com/r/en-U.S./ug1414-vitis-ai/Vitis-AI-Overview

[26] T. Aarrestad, V. Loncar, N. Ghielmetti, M. Pierini, S. Summers, J. Ngadiuba, C. Petersson, H. Linander, Y. Iiyama, G. Di Guglielmo, J. Duarte, P. Harris, D. Rankin, S. Jindariani, K. Pedro, N. Tran, M. Liu, E. Kreinar, Z. Wu, and D. Hoang, "Fast convolutional neural networks on FPGAs with hls4ml," *Mach. Learn., Sci. Technol.*, vol. 2, no. 4, Dec. 2021, Art. no. 045015, doi: 10.1088/2632-2153/ac0ea1.

[27] M. Ulbricht, R. T. Syed, and M. Krstic, "Developing a configurable fault tolerant multicore system for optimized sensor processing," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Noordwijk, Netherlands, Oct. 2019, pp. 1–4.

[28] A. Simevski, O. Schrape, C. Benito, M. Krstic, and M. Andjelkovic, "PISA: Power-robust multiprocessor design for space applications," in *Proc. IEEE 26th Int. Symp. On-Line Test. Robust Syst. Design (IOLTS)*, Napoli, Italy, Jul. 2020, pp. 1–6.

[29] L. Sterpone, N. Battezzati, and V. Ferlet-Cavrois, "Analysis of SET propagation in flash-based FPGAs by means of electrical pulse injection," *IEEE Trans. Nucl. Sci.*, vol. 57, no. 4, pp. 1820–1826, Aug. 2010.

[30] C. De Sio, S. Azimi, L. Sterpone, and B. Du, "Analyzing radiation-induced transient errors on SRAM-based FPGAs by propagation of broadening effect," *IEEE Access*, vol. 7, pp. 140182–140189, 2019, doi: 10.1109/ACCESS.2019.2915136.

[31] C. De Sio, S. Azimi, and L. Sterpone, "On the evaluation of the PIPB effect within SRAM-based FPGAs," in *Proc. IEEE Eur. Test Symp. (ETS)*, May 2019, pp. 1–2, doi: 10.1109/ETS.2019.8791527.

[32] A. Amouri, F. Bruguier, S. Kiamehr, P. Benoit, L. Torres, and M. Tahoori, "Aging effects in FPGAs: An experimental analysis," in *Proc. 24th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2014, pp. 1–4.

[33] A. Amouri and M. Tahoori, "High-level aging estimation for FPGA-mapped designs," in *Proc. 22nd Int. Conf. Field Program. Log. Appl. (FPL)*, Aug. 2012, pp. 284–291.

[34] *Xilinx Power Estimator User Guide (UG440)*. Accessed: Mar. 1, 2023. [Online]. Available: https://docs.xilinx.com/r/en-U.S./ug440-xilinx-power-estimator

[35] Y. Zhang, J. Roivainen, and A. Mammela, "Clock-gating in FPGAs: A novel and comparative evaluation," in *Proc. 9th EUROMICRO Conf. Digit. Syst. Design (DSD)*, Aug. 2006, pp. 584–590.

[36] R. T. Syed, M. Ulbricht, K. Piotrowski, and M. Krstic, "Fault resilience analysis of quantized deep neural networks," in *Proc. IEEE 32nd Int. Conf. Microelectron. (MIEL)*, Nis, Serbia, Sep. 2021, pp. 275–279, doi: 10.1109/MIEL52794.2021.9569094.

[37] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "14.5 Envision: A 0.26-to-10 TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28 nm FDSOI," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, San Francisco, CA, USA, Feb. 2017, pp. 246–247, doi: 10.1109/ISSCC.2017.7870353.

[38] Y. Zhang and Q. Yang, "A survey on multi-task learning," 2017, *arXiv:1707.08114*.

[39] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *Proc. Int. Conf. Eng. Technol. (ICET)*, Aug. 2017, pp. 1–6.

[40] B. Moons, K. Goetschalckx, N. Van Berckelaer, and M. Verhelst, "Minimum energy quantized neural networks," in *Proc. 51st Asilomar Conf. Signals, Syst., Comput.*, Pacific Grove, CA, USA, Oct. 2017, pp. 1921–1925, doi: 10.1109/ACSSC.2017.8335699.

[41] Blender. *Free and Open Source 3D Creation Suite*. Accessed: Apr. 6, 2023. [Online]. Available: https://www.blender.org/

[42] Phased Array System Toolbox. (2023). *Phased Array System Toolbox Provides Algorithms and Tools for the Design, Simulation, and Analysis of Phased Array Signal Processing Systems*. Accessed: Apr. 6, 2023. [Online]. Available: https://de.mathworks.com/products/phased-array.html

[43] T. Zhou and D. Tao, "Greedy bilateral sketch, completion & smoothing," in *Proc. 16th Int. Conf. Artif. Intell. Statist. Mach. Learn. Res.*, vol. 31, 2013, pp. 650–658. [Online]. Available: https://proceedings.mlr.press/v31/zhou13b.html

[44] V. Chen, *The Micro-Doppler Effect in RADAR*, 2nd ed. Norwood, MA, USA: Artech House, 2019.

[45] Y. Zhao, V. Sark, M. Krstic, and E. Grass, "Synthetic training data generator for hand gesture recognition based on FMCW RADAR," in *Proc. 23rd Int. Radar Symp. (IRS)*, Gdansk, Poland, Sep. 2022, pp. 463–468, doi: 10.23919/IRS54158.2022.9904997.

[46] Y. Zhao, V. Sark, M. Krstic, and E. Grass, "Novel approach for gesture recognition using mmWave FMCW RADAR," in *Proc. IEEE 95th Veh. Technol. Conf. (VTC-Spring)*, Helsinki, Finland, Jun. 2022, pp. 1–6, doi: 10.1109/VTC2022-Spring54318.2022.9860976.

[47] *Magnitude-Based Weight Pruning*. Accessed: Jan. 27, 2024. [Online]. Available: https://rb.gy/fms383

[48] F. Kastensmidt, *Fault-Tolerance Techniques for SRAM-Based FPGAs*. Cham, Switzerland: Springer, 2006.

[49] *Soft Error Mitigation Controller IP Product Guide*. Accessed: Aug. 15, 2023. [Online]. Available: https://docs.xilinx.com/r/en-U.S./pg036

[50] 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. Exhib. (DATE)*, Mar. 2020, pp. 1241–1246.

[51] 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.

[52] B. Ghavami, M. Sadati, Z. Fang, and L. Shannon, "FitAct: Error resilient deep neural networks via fine-grained post-trainable activation functions," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2022, pp. 1239–1244.

[53] F. Ponzina, M. Peón-Quirós, A. Burg, and D. Atienza, "E2CNNs: Ensembles of convolutional neural networks to improve robustness against memory errors in edge-computing devices," *IEEE Trans. Comput.*, vol. 70, no. 8, pp. 1199–1212, Aug. 2021.

[54] Y. Li, M. Li, B. Luo, Y. Tian, and Q. Xu, "DeepDyve: Dynamic verification for deep neural networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 101–112.

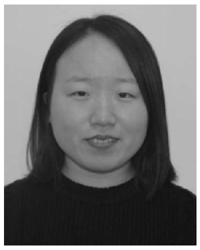
[55] K. Zhao, S. Di, S. Li, X. Liang, Y. Zhai, J. Chen, K. Ouyang, F. Cappello, and Z. Chen, "FT-CNN: Algorithm-based fault tolerance for convolutional neural networks," 2020, *arXiv:2003.12203*.

[56] F. F. dos Santos, L. Draghetti, L. Weigel, L. Carro, P. Navaux, and P. Rech, "Evaluation and mitigation of soft-errors in neural network-based object detection in three GPU architectures," in *Proc. 47th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. Workshops (DSN-W)*, Jun. 2017, pp. 169–176.

[57] F. Libano, "Analyzing and improving the reliability of matrix multiplication and neural networks on FPGAs," Ph.D. dissertation, School Elect. Comput. Energy Eng., Arizona State Univ., Tempe, AZ, USA, 2021. [Online]. Available: https://search.asu.edu/profile/2458369

[58] B. F. Goldstein, V. C. Ferreira, S. Srinivasan, D. Das, A. S. Nery, S. Kundu, and F. M. G. França, "A lightweight error-resiliency mechanism for deep neural networks," in *Proc. 22nd Int. Symp. Quality Electron. Design (ISQED)*, Apr. 2021, pp. 311–316.

RIZWAN TARIQ SYED received the Masters of Science in Communication Engineering (M.S.C.E.) degree from the Technical University of Munich, Germany. Since 2017, he has been with IHP—Leibniz-Institut für Innovative Mikroelektronik, Frankfurt (Oder), Germany, where he is currently a Research Scientist with the System Architectures Department. For the last few years, his work has been mainly focused on fault-tolerant and reconfigurable AI accelerators on FPGAs. He has been involved in many research and development projects at IHP—Leibniz-Institut für Innovative Mikroelektronik (EMPHASE, Space Region, and Open6G Hub).



YANHUA ZHAO received the B.Eng. degree in communication engineering from Qingdao University, China, in 2017, and the M.Sc. degree in information and communication engineering from the Technical University of Darmstadt, Germany, in 2020. In 2020, she joined the IHP—Innovations for High Performance Microelectronics (Leibniz-Institut für Innovative Mikroelektronik), Frankfurt (Oder), Germany, where she is currently a member of the Fault Tolerant Computing Group. Her current research interests include the design of joint communication and sensing systems operating in the millimeter-wave spectrum, FMCW RADAR, CW RADAR, gesture recognition, and vital sign detection.

JUNCHAO CHEN received the Dr.-Ing. degree from the University of Potsdam, Germany, in 2023, on the topic of "A Self-adaptive Resilient Method for Implementing and Managing High-reliability Processing Systems." Since 2018, he has been a member of Prof. Milos Krstic's "Fault-Tolerance Computing" Research Group, IHP—Leibniz-Institut für Innovative Mikroelektronik, Frankfurt (Oder), Germany. Throughout his academic career, he has participated in several international and national research projects. He has published over 20 journal articles and conference papers and has been granted one patent. His research interests include self-adaptive fault-tolerance mechanisms, radiation-induced effects, high-reliability system design, and agile hardware development.

MARKO ANDJELKOVIC received the Dipl.-Ing. degree in electronics from the Faculty of Electronic Engineering, University of Nis, Serbia, in 2008, and the Dr.Ing. degree in computing architectures and fault tolerance from the University of Potsdam, in 2021. Since 2016, he has been with IHP—Leibniz-Institut für Innovative Mikroelektronik, where he is currently a Research Associate with the System Architectures Department. His research interests include characterization and modeling of radiation-induced effects in digital circuits, and rad-hard design.

MARKUS ULBRICHT received the Ph.D. degree from Brandenburg University of Technology Cottbus–Senftenberg, in 2014, on the topic of "systematic lifetime-optimization of highly integrated systems on the basis of nano-structures by means of stress optimization and self-repair." In the following two years, he collected thorough experience as a Test Engineer with Intel Communications GmbH, Munich. To have a stronger focus on his scientific career, he transferred to IHP—Leibniz-Institut für Innovative Mikroelektronik, in 2016, where his first projects involved backend HDL design and the design and implementation of a fault-tolerant radar platform for distance measurements for automated driving. As of 2020, he is leading the fault-tolerant computing group with a strong focus on sensory platforms, open-source hardware, reliable computing, and neuromorphic processing systems.

MILOS KRSTIC received the Dr.-Ing. degree in electronics from Brandenburg University of Technology, Cottbus, Germany, in 2006. Since 2001, he has been with IHP—Leibniz-Institut für Innovative Mikroelektronik, Frankfurt (Oder), Germany, where he leads the Department of System Architectures. Since 2016, he has been a Professor of design and test methodology with the University of Potsdam. He is also leading and coordinating space activities with IHP—Leibniz-Institut für Innovative Mikroelektronik. For the last few years, his work has mainly focused on fault-tolerant architectures and design methodologies for digital systems integration. He has been managing many international and national research and development projects. He has published more than 250 journal articles and conference papers and registered 12 patents.