ITUJournal
Future and evolving technologies

ITU Journal on Future and Evolving Technologies, Volume 6, Issue 2, June 2025

Article

Decision-driven fault-tolerant architecture for vision transformers with real-time error mitigation

Indhuja Gudluru[1], Chunyuan Shen[1], Ke Wang[1]

[1] University of North Carolina at Charlotte, USA

Corresponding author: Chunyuan Shen, cshen6@charlotte.edu

Vision Transformers (ViTs) have evolved in the field of computer vision by transitioning traditional Convolutional Neural Networks (CNNs) into attention-based architectures. This architecture processes input images as sequences of patches. ViTs achieve enhanced performance in many tasks such as image classification and object detection due to their ability to capture global dependencies within input data. While their software implementations are widely adopted, deploying ViTs on hardware introduces several challenges. These include fault tolerance in the presence of hardware failures, real-time reliability, and high computational requirements. Permanent faults that are in processing elements, interconnections, or memory subsystems lead to incorrect computations and degrading system performance. This paper proposes a fault-tolerant hardware implementation of ViTs to overcome these challenges. This hardware implementation integrates real-time fault detection and recovery mechanisms. The architecture includes four primary units: patch embedding, encoder, decoder, and Multi Layer Perceptron (MLP) which are supported by fault-tolerant components such as lightweight recompute units, a centralized Built-In Self-Test (BIST), and a learning-based decision-making system using machine learning model 'decision tree'. These units are interconnected through a centralized global buffer for efficient data transfer, ensuring seamless operation even under fault conditions.

Keywords: Decision tree, fault tolerance, hardware Accelerator, vision transformers

1. INTRODUCTION

Vision Transformers (ViTs) have been widely adopted for different computer vision jobs by virtue of the explosive developments in deep learning [1, 2, 3, 4, 5]. Unlike Convolutional Neural Networks (CNNs) [6], which extract local characteristics from spatially limited convolutional filters, ViTs split an image into non-overlapping patches and treat them as independent tokens. These patches are then run via an embedding layer and handled by a sequence of Multi-Head Self-Attention (MHSA) [7] and Feedforward Network (FFN) [8] layers. This architectural change lets ViTs record local and global dependencies inside an image, thereby improving classification, object identification, and segmentation tasks' performance. Nevertheless, in spite of their benefits, ViTs need a great processing capability, while considering the complexity of self-attention processes, which performs significant matrix multiplications over several layers. Therefore, hardware accelerators are used to promote parallelism, optimize memory access, and raise execution efficiency [9, 10].

A typical ViT hardware accelerator [11, 12, 13, 14] is featured with processing elements and memory systems to manipulate the complex matrix operations in attention mechanisms. These designs concentrate on parallelism, fast data access, and efficient control, which meets real-time performance requirements. Using specialized Processing Elements (PEs) optimized for tensor computations, ViT hardware accelerators assure fast processing for real-time uses [2]. These accelerators include parallelized architectures which improve throughput, also lowering latency in large-scale vision applications which is dissimilar

from general-purpose CPUs. By reducing duplicate memory access and hence enhancing energy efficiency and computational performance, hardware-aware attention techniques maximize self-attention computations. However, deploying ViTs on hardware accelerators raises serious concerns about reliability, as most designs prioritize speed and efficiency while often overlooking fault tolerance and system stability [15, 16]. As a result, even minor hardware faults in processing elements, memory, or interconnections can lead to misclassifications or system failures, especially in real-time or safety-critical applications. ViTs depend on great parallelism hence any breakdown in memory units, interconnections, or processing elements would cause execution faults, misclassifications, and general system instability [17, 18, 19].

In ViT accelerators, both permanent and transient flaws become main issues as technology scales. Hardware aging, or manufacturing flaws cause permanent errors in memory subsystems, interconnection links, and PEs, hence rendering irrevocable failures [17, 18, 19]. In contrast to transient faults, which are caused due to voltage fluctuations or timing problems and can be fixed with simple retries, permanent faults require vigorous hardware-level detection and correction or mitigation mechanisms to prevent permanent system failures [20, 19]. ViTs rely on complex matrix multiplications and token embedding transformations, so even small faults can propagate across layers which leads to cascading computational faults [15, 16]. These errors affect token embeddings, change attention weight computations, and finally provide erroneous feature representation. Real-time dependability is a major difficulty since ViTs analyze high-dimensional data in a massively parallel way and such faults can cause significant accuracy losses [18, 9].

Sustaining the model performance in practical implementations depends on progressing the effective fault mitigating techniques for ViT hardware accelerators. Error Correction Codes (ECC) [21], Triple Modular Redundancy (TMR) [22], Built-In Self-Test (BIST) [23], and rollback systems have been investigated among several fault-tolerance approaches [22, 23, 9, 10]. Although, ECC-based techniques manage computational errors in PEs and interconnections, the memory-related issues still persist [16, 9]. Although, TMR assures reliability by performing re-computations over several hardware units, its high area overhead and power consumption for edge-based ViT accelerators make it unsuitable for real-time applications [22]. Re-executing calculations from a past state helps rollback approaches reduce errors but they still create high latency, hence they are not suitable for real-time inference [9]. While BIST detect faults inside compute units, traditional systems depend on predefined failure conditions instead of dynamically changing to match real-time fault circumstances [23, 24].

Existing fault-tolerance techniques for ViTs often rely on static redundancy, fixed error thresholds, or retraining-based robustness, which fail to adapt dynamically varying hardware faults in real time. These methods either experience high power and area overhead [22], lack runtime adaptability [21, 23], or are unsuitable for edge deployments due to computational complexity. To address these limitations, this paper proposes a lightweight, interpretable, and computationally-efficient dynamic learning algorithm for vision transformers, offering real-time fault classification and adaptive mitigation [25]. To dynamically allocate fault-mitigating techniques, this design combines PEs, a centralized BIST [23], a lightweight recompute unit, and a learning-based decision logic unit. Fault reports generated by the centralized BIST are forwarded to the decision-making unit, which continuously monitors runtime behavior and dynamically chooses the appropriate mitigation strategy based on observed fault severity.

The permanent faults are addressed in the PE array of ViT through a dynamic, runtime fault classification and mitigation mechanism.For fault classification and dynamic mitigation, a framework is proposed that integrates a learning-based decision logic trained on runtime features. Among these runtime features, relative error emerged as a strong discriminator for severity classification. To support effective classification, two boundary thresholds were selected through hyperparameter tuning, applied specifically to the relative error feature, to distinguish low, medium, and critical faults. These boundaries were used to generate supervision labels for the decision tree, which then learned to classify severity across layers and varying runtime conditions. This dynamic strategy allows the system to selectively trigger recomputation or reallocation only when required, enabling robust and efficient inference even under severe permanent fault conditions, without the need of retraining or extensive computational resources.

The inference performance of the proposed reliable ViT framework using key metrics such as accuracy, loss, execution time, throughput, and energy consumption are evaluated. The accuracy of fault-free ViT simulator is 80.8%. Then, during inference, after the introduction of faults and the application of dynamic fault mitigation strategies, the proposed framework achieved an execution time of 18.03 seconds, and a throughput of 555 images/second. When the model is injected with a high fault injection rate (0.9), the inference accuracy dropped noticeably to 2.15%, the proposed mitigation mechanism restored it to 77.68%. Similarly, the inference accuracy loss increased sharply under fault conditions but the proposed mitigation mechanism reduced it significantly was significantly. Though execution time increased to 25.85 seconds due to recomputation, throughput persisted stable, and energy consumption remained within acceptable bounds. These inference-time results confirm that the

framework ensures robust real-time performance with minimal degradation, even under severe fault conditions.

2. BACKGROUND AND MOTIVATION

Modern deep learning models have demonstrated exceptional capabilities in computer vision tasks, but their deployment on hardware platforms is increasingly constrained by limitations in performance reliability. As models grow deeper and more data-intensive, their execution becomes tightly bound to hardware efficiency and stability. This is especially evident in transformer-based architectures, which rely on parallel computation, memory access coordination, and token interdependencies any disruption in this execution pipeline can cause cascading errors. Ensuring robustness under such conditions is no longer a peripheral requirement but a core design consideration, particularly in environments where fault occurrences, aging effects, and real-time demands coexist. This section discusses the working of vision transformers, their associated hardware design, and the increasing concern around permanent faults. It also highlights the drawbacks of existing solutions and concludes with the justification behind this work's proposed decision-tree based fault mitigation strategy.

2.1 ViT background

Vision transformers, as shown in Fig.1, extract important feature representations from raw pixel data and convert images into patch embeddings through a multi-stage pipeline. Initially, the process splits the input image into non-overlapping fixed-size patches [1]. Each patch is linearly projected into a one-dimensional vector which preserves spatial context efficiently. These embeddings are then mapped into a higher dimensional space, followed by the addition of positional encoding to retain the relative ordering of tokens.

This sequence is then processed by transformer encoder blocks, each comprising Multi-Head Self-Attention (MHSA) [7] and Feedforward Network (FFN) modules [8]. Attention layers [7] assess the relevant features across all token pairs using scaled dot product attention; this allows ViTs to capture global context from the first layer ahead. Feedforward layers further refine the features through non-linear transformations, at the same time the residual connections and normalization layers assure numerical stability and enable smoother gradient propagation [2]. These processed embeddings are stored in buffer units, making them available for downstream decoding or task-specific outputs.

2.2 ViT hardware

Considering the resource-constraint in ViTs, dedicated hardware accelerators [11, 12, 13, 14, 2] are designed to execute their operations effectively. These accelerators are optimized to handle the core operations of transformer design such as complex matrix multiplications, self-attention [7] computations, and token-wise feedforward layers. A standard ViT accelerator includes units such as a patch embedding unit, Processing Element (PE) array [18, 26], global/local memory buffers, control unit, non-linear activation units, and output classifiers.

Patch embedding units [1] convert spatial patches into embedded tokens. These tokens are stored in the buffer and forwarded to a PE array which has been configured to perform matrix multiplications, accumulation, activation functions, and normalization operations. Buffers lower off-chip memory access and simplify the reuse of intermediate embeddings. Non-linear units [1, 27, 28] which apply activation functions like ReLU or GELU improve feature transformation. For target tasks, the output units manage the embeddings. A Network-On-Chip (NoC) [29] or custom interconnected fabric facilitates the rapid and synchronized communication among PEs, memory units, and the control logic. The control unit manages memory access, computational scheduling, and token routing across several transformer layers. These accelerators are highly optimized for performance, at the same time they are also susceptible to hardware faults due to the density of computation, synchronization dependencies, and memory bandwidth constraints [12]. These vulnerabilities raise serious inquires about system dependability and fault tolerance, especially under permanent fault scenarios [15, 16].

2.3 Permanent hardware faults in ViTs accelerators

In ViT hardware accelerators, permanent faults are major obstruction affecting memory subsystems, computational units, and interconnections [18, 17, 30]. In contrast to transient faults [10], which occurred due to brief voltage fluctuations or environmental causes and which can usually be fixed, permanent faults are caused by physical degradation, aging, manufacturing flaws, or continuous device stress and as a result they are irreversible [19, 20]. PEs of hardware-accelerated ViTs run parallel matrix multiplications necessary for self-attention computations. A defect in one PE affects computations across several layers, causing cascading errors that reduce model accuracy [15, 16]. Matrix multiplications and self-attention operations are linked such that mistakes spread and influence token embeddings and attention weights. Faults in interconnections restrict data flow causing computational delays, and the faults in memory subsystems change the

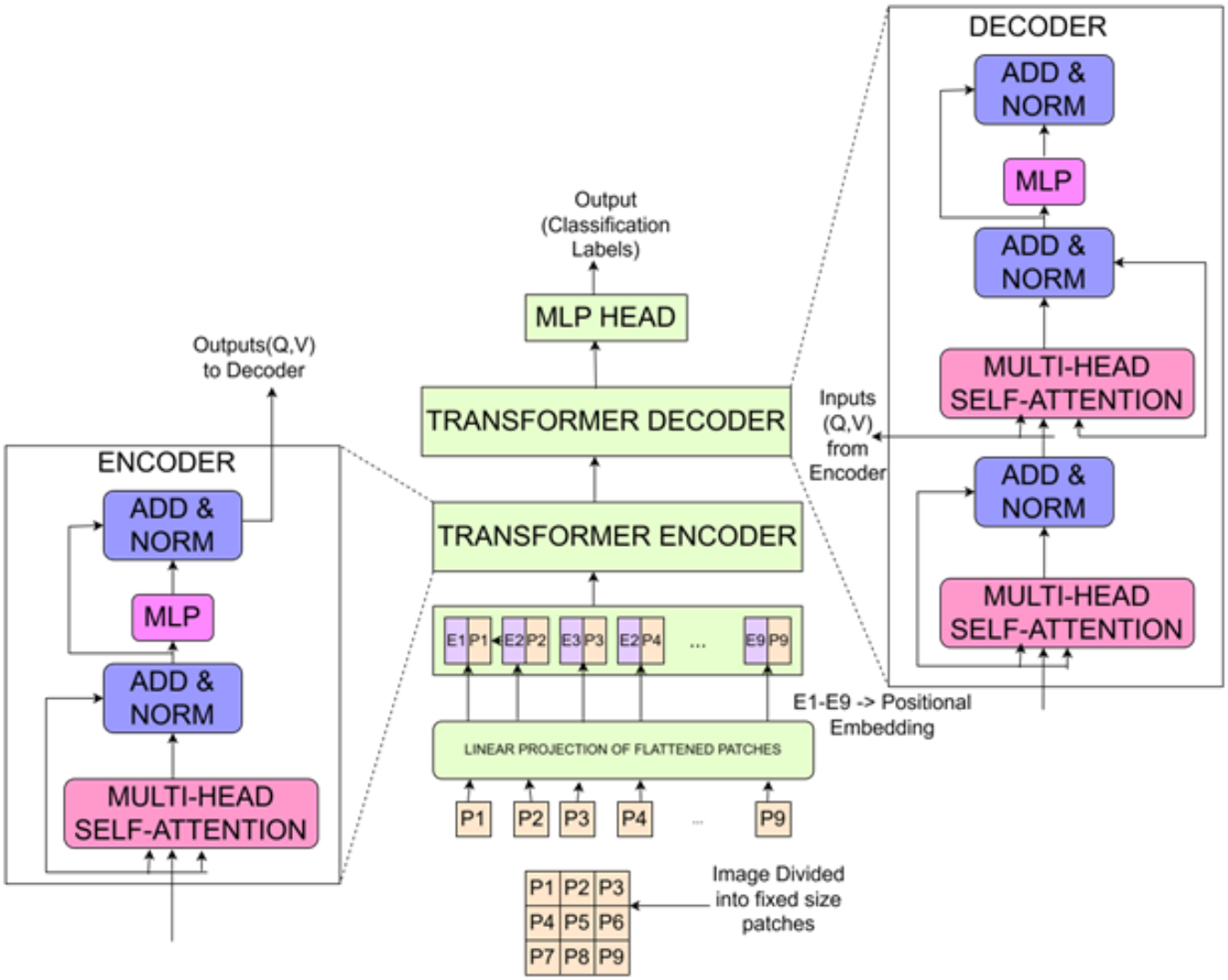


Figure 1 – Vision transformer[1], P1-P9 indicates image patches, E1-E9 indicates positional embeddings

stored representations [9]. ViTs run on high-dimensional data in a massively parallel fashion, hence even a minor fraction of permanently defective PEs can greatly lower model accuracy and general system dependability[18, 15].

2.4 Motivation

Reducing persistent defects in ViT accelerators calls for adaptive techniques able to dynamically find and fix mistakes in real time. Deep learning accelerators have made extensive use of current fault-tolerance methods such as Algorithm-Based Fault Tolerance (ABFT) [15], BIST [23], and Triple Modular Redundancy (TMR) [22], but they have major restrictions. ABFT struggles in non-linear layers, therefore restricting its usefulness in deep systems even if it offers linear error correction [15]. BIST finds flaws but lacks adaptive recovery systems, therefore neglecting the effects of long-term hardware degradation [23]. TMR reduces mistakes by spreading calculations over redundant components; however this method generates substantial power and area overhead and is not appropriate for ViT installations with limited resources [22]. Although they need retraining and cannot adjust to real-time fault circumstances, fault-aware training techniques seek to increase model robustness by exposing the network to simulated failure scenarios [20, 19]. Although pruning and quantization methods lower bit precision sensitivity, hence improving resilience, they do not solve the basic problem of persistent hardware failures [25, 18].

Recent research has explored various ways to reduce faults in deep learning accelerators. Dynamic error mitigation in NoCs indicates a predictive way for error detection, deploying machine learning models to detect and fix faults ahead of execution issues occurring [31]. The severe performance reduction caused by PE failures in systolic designs is highlighted by analyzing and mitigating the

impact of permanent faults on a systolic array-based neural network accelerator and identifies redundancy-aware fault tolerance techniques [18]. Although the higher computational overhead is an bottleneck, algorithmic strategies for sustainable reuse of neural network accelerators provides ways to reuse partially defective hardware by adjusting computational workloads [16]. Fault-aware neural architecture, which examines networks flexible to hardware faults for robust edge accelerators, looks at fault-tolerant architecture that is optimal for edge AI systems [15]. Real-time self-monitoring systems, like Self-healing deep learning accelerators via online monitoring and reconfiguration, dynamically reconfigure processing resources to bypass faulty units [24]. Mitigating the impact of faults in the weight memory of DNN accelerators addresses quantization methods that minimize error propagation in uncertain hardware environments, hence enhancing general model dependability [25]. As vision transformers become extensive in hardware-accelerated environments, it is significant to ensure resilience under permanent hardware faults. Traditional approaches fall short in adaptability, overhead, and real-time execution. This paper directly addresses these gaps. A learning-based fault mitigation framework is proposed, specifically designed for ViT accelerators. Unlike existing work, this is the first approach to handle permanent faults in the PE array using a dynamic, runtime severity classification system. Faults are categorized into critical, medium, and low levels not through static thresholds, but through a learning model trained on fault scenario data and applied live inside the control unit. Based on the predicted severity, the system makes real-time decisions: critical faults trigger the lightweight recompute unit, medium faults are reallocated to healthy PEs, and low-severity faults are tolerated to keep the system efficient. This design eliminates the need for full-scale redundancy or retraining and instead enables fast, interpretable, and fine-grained correction directly in the ViT pipeline. The ability to classify and mitigate permanent PE faults dynamically using the proposed integration of learning-based decision logic is the core novelty of this work, setting it apart from all prior fault-tolerant ViT designs.

3. THE PROPOSED RELIABLE VIT ARCHITECTURE

3.1 The proposed hardware design overview

Our proposed reliable ViT architecture, as shown in Fig. 2, is designed to deliver high performance while dynamically managing hardware faults during inference. We built upon the foundational hardware architecture of existing ViT accelerators such as ViA [11], and MorphGCNX [14], which prioritize throughput, parallelism, and energy efficiency. These designs provide the basic compute infrastructure with Processing Elements (PEs), patch embedding units, and interconnections optimized for matrix operations. However, they lack fault resilience mechanisms that are essential for real-time deployment under permanent hardware degradation.

To address this, the proposed reliable ViT architecture is modified using these baseline architectures by integrating three key fault-tolerant components: a centralized BIST module for real-time fault detection, a learning-based decision logic unit that classifies fault severity using runtime features, and a lightweight recompute unit to selectively correct critical PE-level errors. These enhancements transform a performance-centric accelerator into a reliability-aware system that maintains ViT inference even in the presence of permanent hardware faults. As illustrated in the architectural diagram, the integration is seamlessly embedded into the ViT pipeline, ensuring minimal performance overhead while significantly improving fault tolerance.

The compute unit includes several PEs in an array at the center of this system performs the crucial operations such as matrix multiplications for self-attention, and feedforward transformations. Given that the permanent faults can affect PEs, the erroneous computations are handled by the integrated lightweight unit in the computation unit. This unit recomputes only the incorrect computations. The initial level of processing takes place in the patch embedding unit. Then, the non-linear unit adds positional encodings to patches to guarantee preservation of spatial links. Acting as a shared memory resource, the global buffer stores processed outputs, intermediate embeddings, picture patches, and temporary storage of hardware components which provides seamless communication between several hardware components. BIST mechanisms in the control unit constantly check the PE array to detects the hardware faults. Instead of relying on a real-time adaptive learning mechanism, the architecture uses a pre-trained learning algorithm model that detects flaws and implements fault mitigation depending on pre-analyzed fault patterns. The learning algorithm is pre-trained to make decisions which will be discussed later in Section 4, follows a organized decision-making procedure derived from pre-computed fault analysis. Depending on the degree of the error, once faults are found the system can either reassign calculations to functional PEs or a lightweight recompute unit. This ordered fault recovery system guarantees accuracy of the ViT model without adding too much latency.

This architecture is particularly designed to provide a hardware-level fault resilience accelerator, as well as preserving the computational performance. The ViT model assures that faults do not spread by combining a structured fault detection with a pre-determined fault categorization method and an adaptive fault mitigation mechanism, therefore enabling deep learning inference

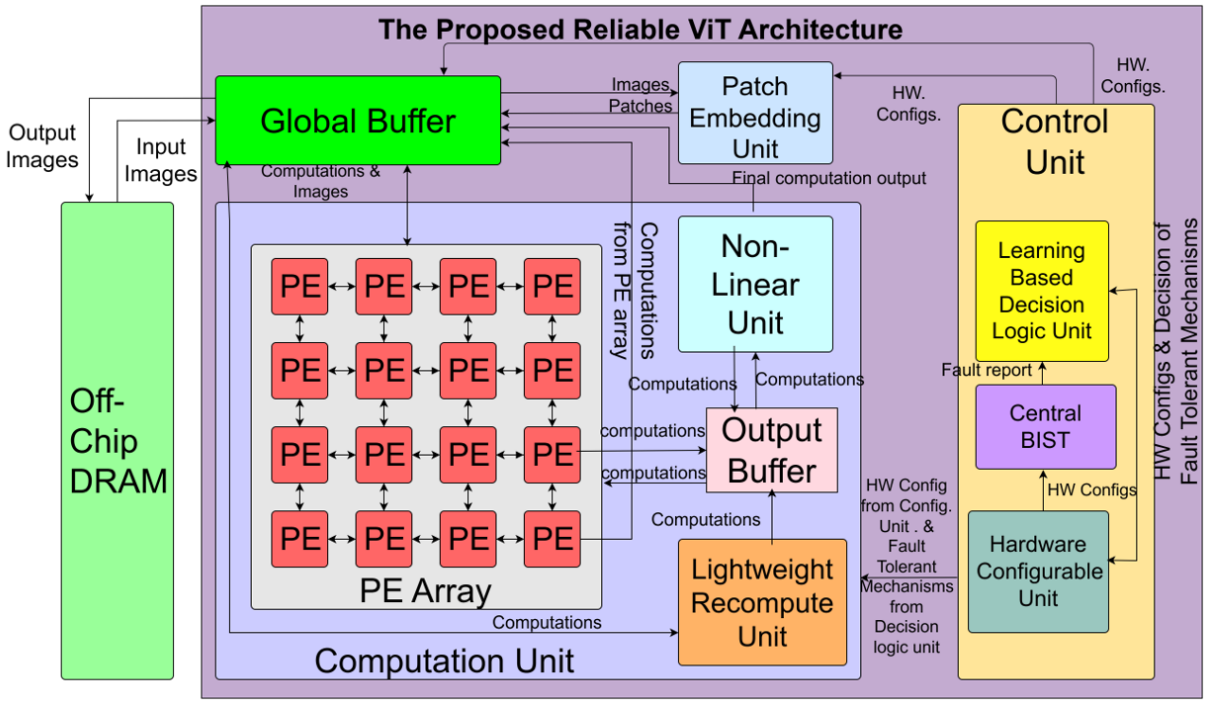


Figure 2 – Proposed reliable ViT architecture

to proceed unconstrained, even under faulty conditions.

3.2 Computation unit

The compute unit forms the main processing engine of the fault-tolerant ViT paradigm. Here, all important transformations, including self-attention, feedforward systems, and feature extraction, occur. Vision transformers largely depend on matrix multiplications and sequential computations, hence the architecture of this unit is designed to efficiently manage high computational loadings.

Inside the computation unit, a PE array [26] executes matrix operations, and other crucial computations which allows the transformer to concurrently process many image patches. Since the hardware faults are making these PEs defective, this causes the PE to output erroneous computations. This causes a greater negative impact in accuracy, so the incorrect computations from faulty PEs have to be controlled. To manage this, the lightweight recompute unit is thereby connected with the PE array. This unit selectively recalculates only the affected operations due to erroneous computations, therefore lowering performance overhead, as well as preserving accuracy.

Then the non-linear unit is used for the activation functions. Transformers rely on activation functions such as GELU or ReLU to increase learning capacity, hence this unit is absolutely essential in enhancing feature representations. These components, PE array, lightweight recompute unit, and non-linear unit, together ensure that the computation unit works consistently even in the presence of hardware faults which are permanent faults at the PE array. Thus, this makes the ViT model exhibit higher fault tolerance than traditional models.

3.2.1 Processing element

The PE array (Fig. 3) forms the core of the compute unit which is responsible for doing necessary matrix multiplications and attention operations. Every PE runs vector multiplications, accumulations, and weight updates acting as a small independent processing core. Vision transformers need large parallel computations, hence these PEs coordinate the processing of several patches in parallel to optimize the model's efficiency. The control unit first ensures that operations follow the correct sequence and directs the flow of data. Each PE consists of several crucial components such has MAC, input and weight buffers, and lightweight BIST. The input buffer stores input data before it is processed at the same time the weight buffer stores learned weights for matrix multiplications. The core computations such as the matrix multiplication of input vectors and stored weights, addition of output of multiplier and partial sums take place in the vector multiply-accumulate (MAC) unit, which aggregates results before forwarding the results to further stages. Since then the hardware defects can occur anywhere, the PE design has a lightweight BIST [23] mechanism. This BIST tracks the computations of the individual PE constantly, identifying the errors when differences are found. The lightweight recompute unit

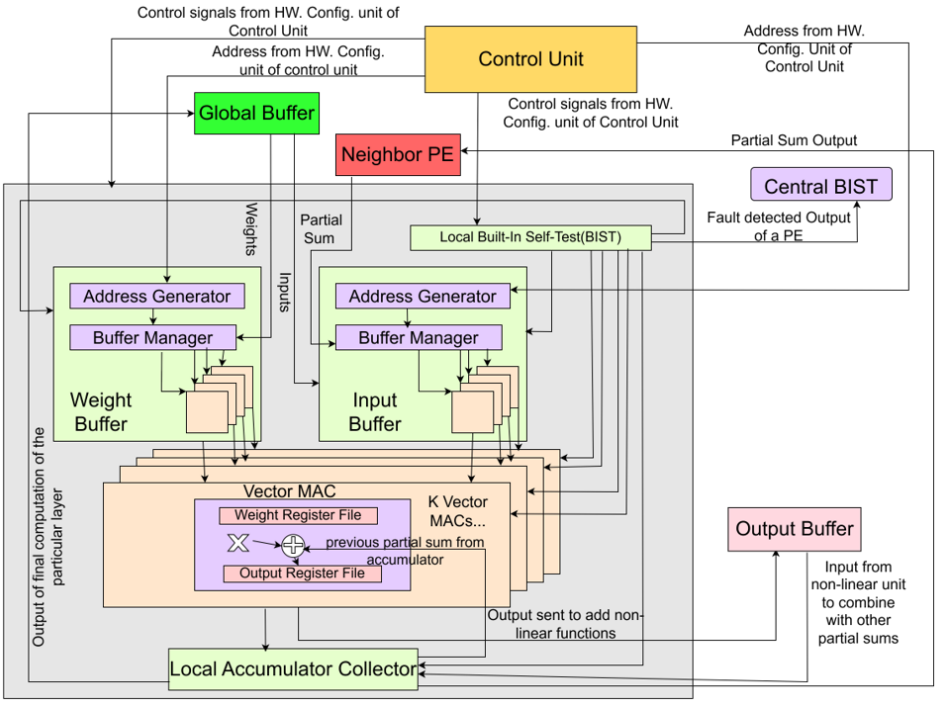


Figure 3 – Processing element architecture

arbitrates to manage recalculations if an incorrect computation is found; therefore, it prevents the circulation of faults throughout the system. The accumulation collector collects the results before they are transferred to the next processing stage and the softmax calculations in the PE ensure that probability distributions are computed for attention mechanisms accurately. The effective and error-free computations are achieved by this well-organized PE design, which guarantees the ViT model to operate as expected, even in the presence of defects.

3.2.2 *Lightweight recompute unit*

Error correction is mostly dependent on the lightweight recompute unit to preserve computational integrity while minimizing extra computational and area overhead. This unit selectively recalculates just the processes impacted by malfunctioning PEs instead of using broad redundancy approaches, which would slow down the system. Control logic unit in the control unit controls the lightweight recompute unit. It sends the control signals when learning-based decision logic decides to do recomputation in the lightweight recomputation unit. The recompute unit gets the matching inputs and weights from the global buffer, runs the required recalculations using its vector MAC unit, and subsequently updates the output to the global buffer with updated values when a PE reports incorrect values through the lightweight BIST. The recompute unit also performs softmax calculations and has an accumulation collector combining corrected outputs with current data. In contrast to individual PEs, this unit only performs MAC operations, softmax computations, and partial sum computations, thereby reducing the computational and area overhead. Thus, this made the recomputation unit lightweight. Also, it runs parallel with other PEs, thus also reducing execution time. Using fault tolerance in a selected and effective way when using this unit preserves accuracy and performance in the ViT model under critical permanent fault conditions.

3.2.3 *Non-linear unit*

Though most of the calculations in ViTs are driven by matrix multiplications and attention processes, the non-linear unit [28], [27] guarantees proper application of activation functions. This unit adds non-linearity to increase the expressiveness and decision-making capability of deep learning models since linear transformations by themselves are not adequate for them. This unit is responsible for applying the activation functions, including Gaussian Error Linear Unit (GELU) or Rectified Linear Unit (ReLU), to the converted embeddings. It then passes the inputs from the PE array through the activation functions prior to transferring the output to the next stage in the pipeline; it passes inputs from the PE array through the activation function. Furthermore, the non-linear unit guarantees that throughout the patch embedding phase positional encodings are applied appropriately. This unit preserves spatial information by including learned positional encodings into the embeddings as transformers lack a built-in concept of spatial structure like CNNs. As this unit mainly handles only the element-wise opera-

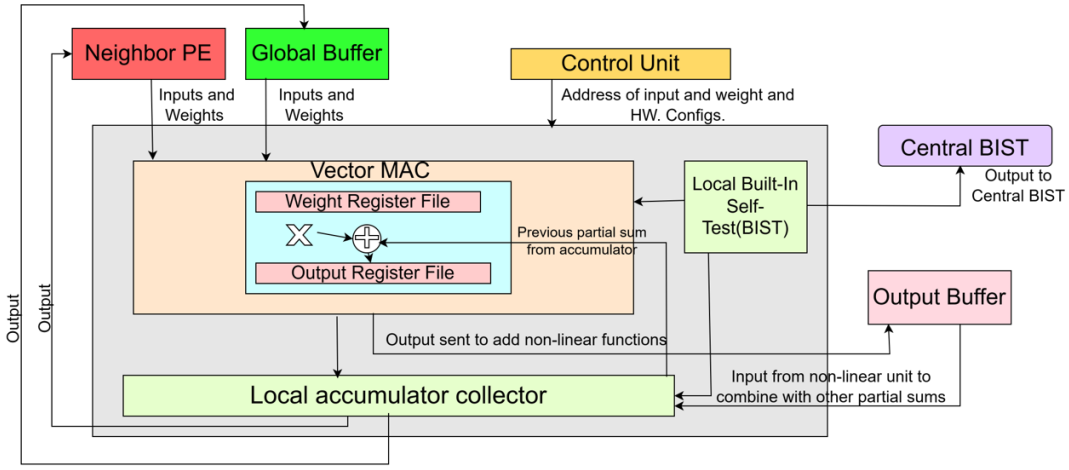


Figure 4 – Lightweight recompute unit architecture

tions, the impact of faults occurring here mostly is lower on the final output compared to faults in the processing element array.

3.3 Patch embedding unit

Initially, the ViT model forwards the input images stored in the global buffer to the patch embedding unit which processes the input image [23]. The patch embedding unit serves as an intermediary between conventional transformers, which run on sequences instead of raw picture data, therefore transforming spatial image characteristics into a format the transformer can effectively handle. This unit guarantees that, while maximizing the input representation for the next calculations, the model preserves important spatial information. The patch divider divides the input image into fixed-size patches, therefore handling the first stage in the patch embedding process. This unit in the ViT model divides the whole image into smaller patches in place of processing them altogether at a time, which makes the model treat every patch as an individual token. This is essential since it lets the transformer examine local characteristics while preserving a global relationship of the whole image.

The patches are first divided and then processed through the linear projection unit where 1D vectors are generated. In the beginning, to generate a relative feature representation, a lesser matrix multiplication is performed by every patch. The transformer architecture does not include built-in techniques to identify spatial correlations in raw image data by itself. This drawback is addressed by the patch embedding unit, which explicitly encodes spatial relations into the patch representations by adding the positional encoding. Each patch is mapped to a higher-dimensional space, which makes the model have a structured numerical representation. The features are extracted from this representation and learn self-attention in further stages. The image patches and their positional embeddings are forwarded to the global buffer which ensures proper data flow throughout the system where it is stored permanently and from there the patches are forwarded to the next stage. Thus, the patch embedding unit makes the ViT be able to manage image tasks by organizing the input data in a way that is suitable for processing them further in the transformer.

3.4 Control unit

The control unit constitutes the backbone of fault detection and mitigation in the fault tolerant ViT architecture, therefore ensuring that all computational activities run without any problems even when the more permanent faults occur. It continuously tracks the compute unit, particularly the Processing Element (PE) errors and arranging appropriate mitigating strategies. By means of interactions with multiple components to perform remedial action upon errors, the control unit regulates data flow as well.

This unit processes numerous inputs to maintain operating stability. Although control instructions from the control unit enable synchronizing of execution, fault reports from the BIST mechanisms in the PEs provide real-time error detection signals. The decision unit determines whether incorrect results should be sent for workload redistribution or recomputation. Thus, the control unit also collects performance logs and fault classification reports. After fault diagnosis and classification, the control unit provides outputs to ensure system functionality. These consist of fault mitigation signals sent to the lightweight recompute unit for recalculating erroneous computations or reallocation instructions by offloading work from defective PEs to functional ones, and execution control signals to maintain the overall processing stability. Through coordinated operation, the

control unit minimizes the influence of errors, which helps in optimizing computational efficiency.

3.4.1 Learning-based decision logic unit

Based on fault severity level produced by the central BIST, the decision unit chooses suitable fault-mitigating measures. This unit uses a pre-trained decision tree model, which follows set, predetermined criteria to decide the optimal corrective response for every detected faults The central BIST fault report, which has a fault severity level, supplies the inputs to the decision unit. The decision unit decides how to handle faults depending on severity level rather than examining actual fault data. Once a fault is categorized, the decision unit creates outputs guiding the control logic unit toward the proper path of action. If the critical faults are detected, it alerts the lightweight recompute unit to manage recalculations or guides computation to dual-mode PEs in cases of limited resources. If the fault severity level is medium, the decision unit tells the control logic unit to replace defective PEs with healthy ones, hence preserving processing efficiency free from extra computational load. Low-severity errors allow the decision unit to give bypass commands, therefore enabling calculations to go forward free from corrections. This unit guarantees consistent and effective fault mitigating judgments by depending on a disciplined pre-trained decision tree model. This methodology reduces processing cost while keeping a disciplined approach to fault recovery, hence eliminating the demand for real-time adaptive learning.

3.4.2 Central Built-In Self Test (BIST)

Global fault monitoring inside the ViT model is achieved by the central BIST unit. In contrast to the local error detection by individual BIST mechanisms in each PE, the central BIST offers a system-wide perspective, failure pattern analysis, and general hardware dependability assessment [23, 24]. The central BIST receives inputs from execution logs supplied by the control logic unit, and fault flags from the individual PEs. The central BIST then combines the data from several processing units which helps to identify more general fault trends. Once fault patterns are found, the central BIST generates outputs to classify fault severity level. It reports faults to the decision unit, allowing pre-trained techniques to enable fault mitigating action based on the fault report. The central BIST also gives the control logic unit diagnostic feedback so it may modify processing strategies. If any PEs show consistent failures, the central BIST can label them as unreliable and cause the system to assign their computing responsibilities to functional PEs. The core BIST guarantees efficient management of faults by constant analysis of fault behavior and support of informed decision-making, therefore preventing computational failures from spreading over the system.

3.4.3 Control logic unit

This unit acts as the central processing core which explicates the incoming signals and coordinates the tasks to execute in a proper and correct order. It tracks system conditions continuously and coordinates reactions depending on needs for fault identification and mitigation. The operational integrity is maintained by combining the inputs from several sources. The BIST unit reports faults and their locations; the task execution signals from the computation unit and global buffer to ensure the coordinated data flow; and the decision unit produces fault mitigation decisions. After processing these inputs, the control logic unit offers several outputs from control unit to other units in ViT architecture. Error-handling instructions set mitigating measures when errors are detected using BIST and mitigation processes from the decision logic unit at the same time the task scheduling signals are generated to drive functional PEs which makes them perform specified calculations. It guarantees that processing is continuously performed by always adjusting the execution needs to the fault situations, hence preserving computational stability.

4. THE PROPOSED LEARNING-BASED REAL TIME ERROR MITIGATION ALGORITHM

The proposed fault-tolerant ViT framework is developed to identify, analyze, and mitigate hardware-induced faults in real time while maintaining both model accuracy and computational efficiency. To achieve this, the framework consists of a set of interconnected modules that simulate hardware behavior, analyze fault impact, and adaptively apply mitigation strategies. Key components include a fault injection module that emulates realistic hardware failures, a fault analysis engine that evaluates the impact of those faults, and a decision-making unit that dynamically adapts the model's behavior based on the severity of the faults. The main goal is to ensure the model remains reliable under fault conditions while minimizing the computational burden typically introduced by mitigation techniques.

To mimic hardware failures in the PEs of the ViT at runtime, faults are injected into specific layers and attention heads where computations are typically handled by hardware PEs. These faults emulate real-world hardware failure scenarios such as stuck-at-zero/one conditions, random bit-level corruptions, and distorted matrix operations. This simulation modifies the underlying weights or intermediate computations in a controlled and tar-

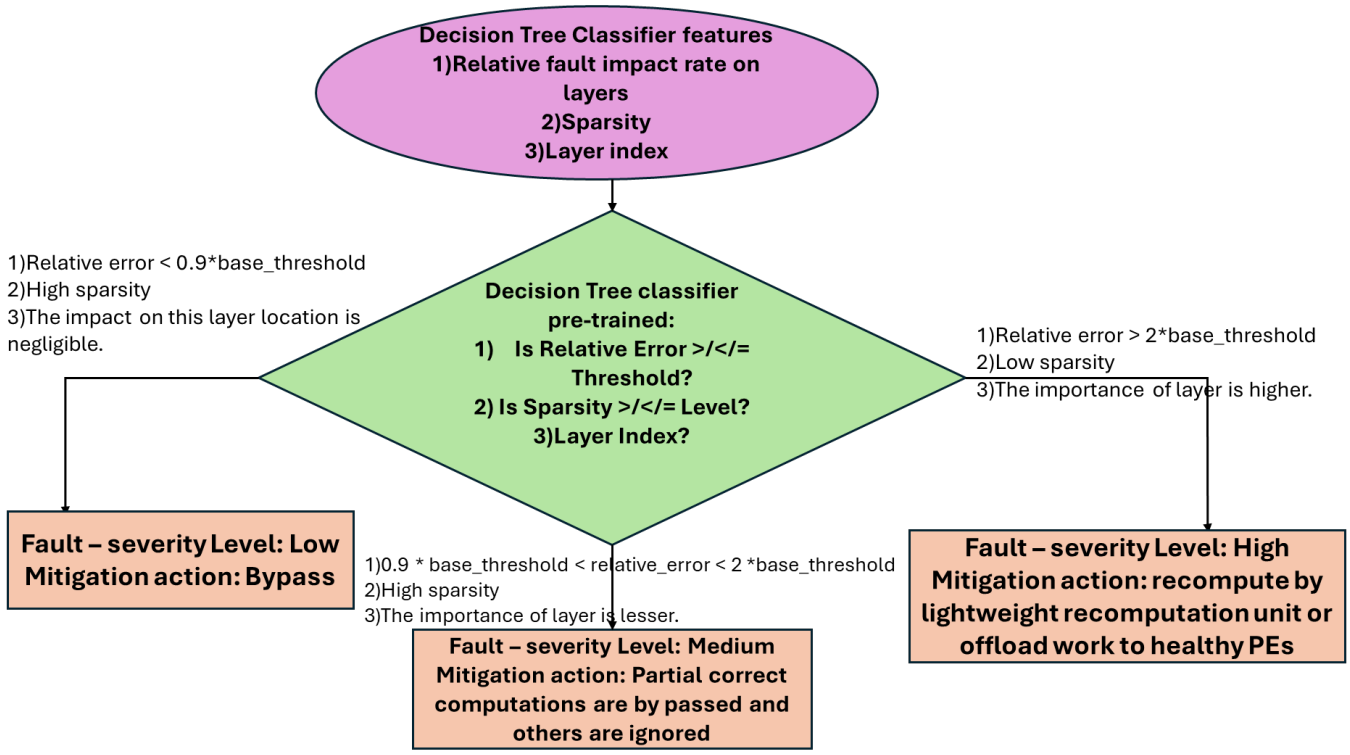


Figure 5 – Example of trained decision tree diagram, base_threshold = how much the faults in the particular layer are impacting the output

geted manner to represent how physical hardware faults would impact the inference pipeline. Following this, a Built-In Self-Test (BIST) module [23] detects the faults and generates a detailed fault report, specifying the affected locations and an initial estimation of severity.

The fault report is then passed to a learning-based decision logic unit, which classifies each fault as low, medium, or critical based on runtime indicators such as relative error, sparsity, and layer index. To support this classification, threshold boundaries were selected through hyperparameter tuning specifically for the relative error input, enabling the system to distinguish different severity levels effectively. Based on the classification, the model dynamically decides how to respond triggering full recomputation only for critical faults, reallocating computation in medium cases, and bypassing negligible errors entirely. This adaptive mechanism avoids the inefficiencies of static redundancy by correcting only when necessary. The system is thus able to mirror real hardware fault behavior while intelligently responding to it in software, ensuring high fault tolerance with minimal performance trade-off.

4.1 Machine learning-based fault classification using decision trees

A critical element of the proposed fault-tolerant ViT framework is its integration of a decision tree classifier for real-time fault severity classification. Decision trees are a class of supervised learning algorithms used for classification and regression tasks. They operate by recursively partitioning the input space according to the feature thresholds by forming a tree-like structure of internal decision nodes and terminal leaves. At each internal node, a feature is selected to split the dataset in a way that maximizes information gain or reduces impurity metrics such as Gini index or entropy. The resulting tree makes predictions by traversing from the root to a leaf node according to the input features. Their interpret-ability, low computational footprint, and fast inference times make decision trees particularly suitable for embedded systems and hardware-level integration.

In the context of real-time fault classification for ViT accelerators, a machine learning model which is decision trees offers a compelling trade-off between model complexity and interpretability. In contrast to several other complex models, such as neural networks or ensemble methods, decision trees do not require deep computation for training or inference and can be easily embedded into the control unit of the accelerator. Additionally, the decision paths ensure their predictable behavior. This is a significant property in fault-tolerant systems. Since ViT faults are sparse and structured, a lightweight model that can map input fault patterns to severity labels with minimal overhead is preferable, and henceforth, decision trees are ideally suitable for this role.

In this research work, this decision tree classifier is trained

to learn the mapping between observed fault characteristics and severity classes such as low, medium, and critical. The classifier runs on a set of features which are received from the fault detection module and these features are specifically selected to reflect the impact of faults within the ViT pipeline. Each input sample to the decision tree consists of three features: relative error, layer index, and fault count. Relative error is computed by comparing the activation output of the faulty model against the corresponding output from a fault-free baseline model during offline simulations. This feature captures how much the output has deviated due to the presence of faults and plays the most important role in determining severity. Layer index provides structural context by indicating the location of the fault in the ViT pipeline, and fault count gives insight into how densely faults are occurring in a particular processing element.

For the severity classification, the relative error is not interpreted directly as a raw value. Instead, it is evaluated against two threshold boundaries: 0.9 times the base threshold and 2 times the base threshold, as shown in Fig. 5. These boundaries were chosen through hyperparameter tuning during training. After testing several values, the (0.9, 2) combination consistently offered meaningful separation between low, medium, and critical severity classes, helping the decision tree learn precise decision boundaries. The base threshold itself is not fixed; it is determined based on the relative importance of each layer. The layer importance values were assigned to capture the varying influence of different layers within the ViT architecture. Since earlier layers deal with the basic embedding and feature extraction, while deeper layers handle higher-level semantic understandings and final decision-making, the impact of faults can vary depending on where they occur. To reflect this, a linearly increasing set of importance scores was assigned across the layers, starting from the initial embedding stages and increasing toward the important layers which carry out computations such as attention, and also this reflects the core computations carried out by PEs in hardware architecture. The chosen range provided a smooth and meaningful distinction between less and high critical layers, helping the model treat the same error differently depending on its location. This approach allowed the fault classification to be context-aware and helped the decision tree apply severity boundaries more intelligently based on where the fault occurred, rather than treating all layers equally. These classification results directly characterize mitigation mechanisms, such as for critical faults; the lightweight recompute unit is used to regenerate error-free activations; medium-level faults are partially corrected through weighted reconstruction techniques; and low-severity faults are bypassed to avoid unnecessary computational overhead. The hyperparameter tuning using a GridSearchCV method along with cross-validation is incorporated in the decision tree classifier, which improved the performance of the classifier to a greater extent. Decision tree parameters such as maximum tree depth, minimum samples per split, and splitting criteria (Ginivs.entropy) were accurately examined which achieved an ideal balance between accuracy and efficiency. The model maintains robustness in spite of various fault scenarios and does not overfit to the training set which all are proved by the evaluation metrics. Evaluation metrics, such as classification accuracy, precision, recall, and F1-score, also proved high performance with very minimal variation between training and validation results. The confusion matrix analysis proved that the classifier constantly identified critical faults with high precision. This analysis ensured minimizing the risk of miscalculating high-severity errors. The framework accomplished a scalable, low-latency proposal for runtime fault response by incorporating this decision tree classifier into the fault classification and mitigation pipeline. Thus, this framework significantly improved the resilience of the ViT architecture by validating intelligent, real-time fault severity estimation and mitigation with minimal computational overhead.

4.2 Training and validation framework

The training and validation framework ensures that the fault tolerance mechanisms are precisely evaluated under various hardware fault conditions. Initially, the system sets up a baseline performance by training the ViT model without faults to measure its accuracy and computational efficiency. Once a stable performance benchmark is set, fault injection is introduced layer-wise in the ViT model to observe the impact of real-time hardware failures in PEs on model behavior. The fault-injected model is then trained across multiple epochs and detected using a fault detection module which replicated a BIST unit in proposed ViT architecture, and different mitigation strategies are applied to refine fault mitigation mechanisms using a learning- based algorithm which replicated the learning-based decision logic unit in the control unit of proposed ViT 2.

During training, the system constantly tracks key performance metrics, such as accuracy, loss, execution time, throughput, energy consumption, and retransmission rates; in both circumstances with and without mitigation mechanisms applied. This dynamic adaptation of the proposed framework is achieved by continuous learning process during training which improved the model's ability to recover from faults while maintaining efficiency. Validation is performed on separate datasets where faults are injected, and the system's fault mitigation mechanisms are evaluated in both scenarios with and without mitigation mechanisms applied. The performance of the fault-tolerant ViT model is compared against the baseline to get a benchmark of the applied fault mitigation strategies.

The analysis is concluded in a layer-wise way to explore in detail the impact of faults at various stages of the ViT pipeline. This provides a detailed evaluation of fault propagation and offers insights into which layers are highly sensitive to hardware-induced errors. The system refines its fault mitigation mechanisms for future iterations by recording layer-wise relative errors and the severity levels classification.

4.3 The working of the proposed framework

The system incorporates a framework, which is designed to apply real-time fault mitigation mechanisms while maintaining computational efficiency. The framework maintains two occurrences of the ViT model: a fault-free reference model that represents as a benchmark for error-free activations and a fault-injected model that runs under various hardware-induced failures. The system detects variations and classifies the severity of computational errors at each layer by constantly comparing the outputs of these two models.

Fault mitigation is adaptively managed according to the fault severity classification. Low-severity errors are bypassed as they won't impact the model's accuracy, which reduces the usage of computational resources. Medium-severity errors go through partial recomputation, where a weighted combination of faulty and correct value of activations is used to fix the accuracy. If the faults are classified as critical-severity faults, the system incorporates the usage of a lightweight recompute unit, which particularly recomputes the impacted activations using stored historical data. This ensures that severe faults are not further forwarded through the other layers of the model, as well as avoiding the unnecessary computational overhead.

Furthermore, the model dynamically incorporates the fault mitigation mechanism in the specific layers or heads that exhibit frequent errors, and are dynamically adjusted based on past learned fault patterns. This allows the system to give importance to the high-impact layers in the fault mitigation process. This ensures the critical components of the ViT model to receive the most computational resources during fault recovery.

4.3.1 Fault injection

To simulate real-world permanent hardware failures, the simulator incorporated a dedicated fault injection module that focuses only on the computations carried out by PEs within the ViT. These include the core operations like multiplications, additions, and attention-related calculations, which are basically computed within each layer's arithmetic data path. Faults are not applied to memory units, control logic, or non-computational modules which guarantees that the simulation mainly focuses on the hardware regions that are prone to permanent failure. Faults are injected during runtime and remain fixed across the entire inference process, replicating the behavior of actual hardware defects such as transistor degradation, wear-out, and stuck-at faults. These faults are not applied uniformly but are distributed across layers and attention heads based on a probabilistic model. This model is designed to reflect the non-uniform degradation observed in real hardware. When a fault is injected into a PE, its output behavior is modified based on the selected fault type, and this faulty behavior persists consistently during model execution, just like a real permanent hardware defect. This fault injection module introduced realistic faults that altered the computational path, ensuring that the simulation closely reflects the impact of real permanent hardware failures. The fault is injected layer-wise to ensure that the fault mitigation is applied in each layer before the faults are propagated to other layers and collapse the entire network. Thus, the framework provides a real-time hardware-aware fault simulation environment for evaluating the effectiveness of the proposed classification and mitigation strategies.

4.3.2 Fault detection

As discussed in section 3, the fault detection mechanism used in this paper is inspired by traditional BIST architectures, such as the design proposed by Wu et al. [[23]]. The core idea of BIST is adapted and replicated in simulation, coordinating with the architectural constraints of a simulated ViT accelerator environment. The BIST mechanism is implemented in the simulation as a lightweight fault detection module that compares outputs from faulty PEs to a known correct baseline (the fault-free model) during inference. Rather than using scan chains or signature registers like in hardware, the software BIST calculates relative error between faulty and reference activations and evaluates activation sparsity, both of which act as indicators of faulty behavior.

The system uses local BIST and central BIST which is similar to the one in the proposed hardware ViT architecture. Local BIST operates at the level of individual PEs and is responsible for immediate, low-overhead detection of abnormal behavior. It flags a PE as faulty based on its output characteristics during execution, such as anomalous error levels or unchanging outputs. Once a fault is detected by the local BIST, that PE is permanently marked as faulty for the remainder of the run, which reduced computational overhead.

In contrast, the central BIST serves as a global collection and coordination module. It gathered the fault reports from all local BIST units and generated a structured fault

report, which includes fault location, relative error magnitude, activation sparsity, and layer-level fault density. This global report is then passed to the fault analysis module and then it is used by the decision tree classifier for fault severity classification. This implementation preserves the low-latency and localized detection philosophy of hardware BIST, while being fully integrated into the software simulation framework, providing a reliable and efficient mechanism for fault diagnosis in the proposed system.

4.3.3 Fault classification and decision mechanism

The fault classification and mitigation mechanisms are carried by the decision tree classifier, as discussed in the earlier section of this section. The framework analyzed dynamically categorizes faults into low, medium, or critical severity levels based on these extracted features by the decision tree classifier. Then the decision tree classifier decides what action to apply to each severity level. As discussed earlier, the fault mitigation mechanisms are applied. Low severity faults have minimal impact on accuracy and so they are bypassed, as discussed earlier, while medium severity faults have noticeable impact. The full recomputation is not done and allowing the faulty output to propagate only when it stays within a tolerable relative error range, as detected by the decision tree classifier. No explicit correction is applied instead, the model treats the medium severity outputs as passable, allowing inference to continue without additional recompute logic or triggering a fallback. Critical faults essentially reduces the model's outputs and so addressed using more aggressive mitigation strategies such as recomputation or offload work to healthy PEs. This learning-based method makes the system dynamically adapt under various fault conditions and refine its decision boundaries with continued training.

5. PERFORMANCE AND EVALUATION

The performance of the fault-tolerant ViT architecture is analyzed under four conditions: fault-free architecture where no faults are injected, faults injected but the application of fault mitigation strategies are not applied and the proposed dynamic fault mitigation and static fault mitigation are applied. The evaluation includes classification accuracy, loss, and computational efficiency across different fault injection rates. Each figure highlights a specific aspect of the model's behavior under varying fault conditions.

5.1 Evaluation setup

The proposed fault-tolerant ViT framework is precisely evaluated by an extensive simulation environment which was developed using PyTorch and associated libraries. The implementation integrates fault injection, permanent faults feature recording and a learning-based mitigation module within the transformer execution pipeline. The framework is designed to simulate real-time hardware-level failures and evaluate their impact on model performance, execution time, energy consumption, and classification reliability under different error conditions. The base model used in all experiments is a pretrained vit_base_patch16_224 obtained through the timm library. Further, this model is duplicated internally into two instances, one is the ideal "healthy" reference model and the other as the fault-prone "faulty" model into which faults are actively injected. The healthy model is frozen to provide error free and correct values of activations for providing a benchmark to the proposed work. Faults are introduced through a custom-built PE fault system that injects permanent errors into selected Query-Key-Value (QKV) weights of the attention heads across 12 transformer layers. Each head within a layer has a stochastic fault assignment, with fault types including stuck-at-0, stuck-at-1, multiplier corruption, adder errors, and logical distortions. The ImageNet-1K training dataset is used for training and evaluation which is processed using standard ViT-compatible transformations such as resizing, cropping, normalization, and random augmentation for effective feature extraction and improved model performance. Then, the ImageNet-1K validation dataset is used for inference. A standard 80:20 train-validation split is applied. The batch sizes are set to 64 with pinned memory and multiple worker threads used to maximize throughput.

For classification under various fault conditions, a lightweight decision tree classifier is trained offline and integrated into the runtime control logic. It uses three core features extracted during fault analysis: relative error magnitude, activation sparsity, and layer index. These features are collected during execution by comparing faulty and healthy layer-wise activations and are used to train the classifier using GridSearchCV for hyperparameter tuning. The fault severity levels are labeled as low, medium, and critical. The training and validation pipeline is managed using a fault tolerance trainer class which encapsulates epoch-based execution, real-time severity estimation, dynamic fault mitigation logic, and metric logging. Performance is evaluated across five error rates: 0.1, 0.3, 0.5, 0.7, and 0.9. At each error rate, the following performance parameters are analyzed in detail: accuracy, loss, throughput (images per second), execution time, energy consumption, retransmission rate, and layer-wise relative error trends. In addition, performance of decision tree classification is evaluated using accuracy, precision, recall, and F1-score. These metrics collectively

validate the resilience and efficiency of the proposed fault-tolerant architecture under high fault stress conditions. The following sections show the outcomes of the performance metrics of the proposed framework, including accuracy, loss, throughput, execution time, and energy consumption during inference of the model and the performance metrics of the decision tree.

5.2 ViT inference accuracy and loss analysis

The graph below in Fig. 6 illustrates the inference accuracy of the fault-tolerant ViT model under different fault injection rates. The x-axis represents the fault injection rate, while the y-axis indicates accuracy percentage.

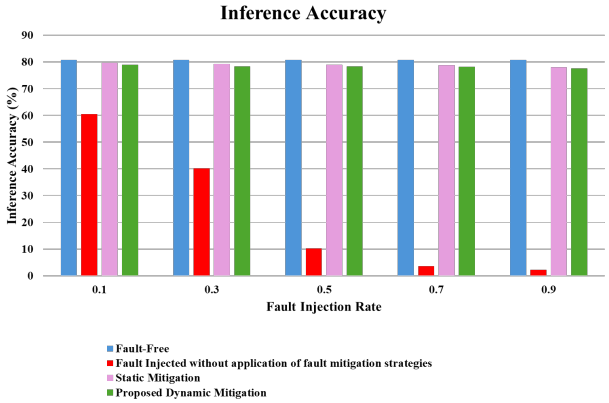


Figure 6 – Impact of fault injection and mitigation on inference accuracy

In the baseline model, the system constantly achieves high accuracy across all fault injection rates, maintaining 80.8%. This ensures that the model performs as expected without any computational faults. However, when faults are injected, accuracy declines sharply. As the fault injection rate increases, the drop is higher, with accuracy decreasing to 2.15%. This highlights the essential impact of permanent faults in Processing Elements (PEs), severely decreasing the classification performance. After applying fault mitigation strategies, performance of the model is improved. Static mitigation restored accuracy but exhibited high overhead, which will be discussed in the further upcoming sections. In contrast, dynamic mitigation restored accuracy restored efficiently with 77.68% at a 0.9 fault injection rate. The baseline levels are not fully recovered but the mitigation successfully prevents major failure and maintains a robust level of classification performance, even in the presence of high fault injection rates.

The graph below in Fig. 7 presents inference loss under different fault conditions. The x-axis represents the fault injection rate, while the y-axis shows the loss value. The loss is analyzed under four conditions: baseline (no faults), faulty (with faults), static mitigation which is the model after the static mitigation mechanism has been applied to the model, and dynamic mitigation which is the model after dynamic mitigation mechanisms have been applied to the model.

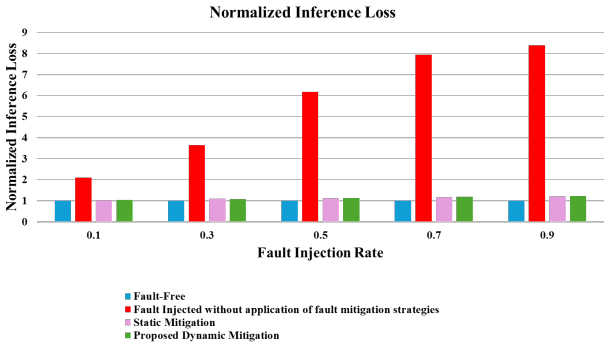


Figure 7 – Impact of fault injection and mitigation on inference accuracy loss

Loss is computed using the cross-entropy loss function between the model's predicted output logits and the ground-truth class labels for each batch. In the baseline condition, loss remains low across all fault injection rates, ensuring the stability of model performance. However, when faults are introduced, loss increases sharply under faulty conditions, particularly at higher fault injection rates (0.9), where the loss exceeds 6%. This suggests that the model struggles to conclude properly when faults impact the computations. After applying fault mitigation mechanisms, static mitigation and dynamic mitigation behave differently. Static mitigation exhibited a spike in loss as the fault rate increases. Here, due to computational overhead and inefficiencies due to unwanted corrections, the loss peaked at high fault rates. In comparison, dynamic mitigation demonstrates stability, maintaining a relatively low and consistent loss across all fault injection rates. This indicates its effectiveness in selectively correcting only critical errors, preserving computational efficiency and model reliability.

Overall, dynamic mitigation outperforms static mitigation in terms of maintaining lower loss under high fault conditions, while the faulty system where no mitigation mechanisms are applied performs worst.

5.3 ViT inference throughput analysis

The graph below in Fig. 8 presents the inference throughput (images processed per second). The x-axis represents the fault-injection rate, while the y-axis indicates normalized throughput.

In the baseline condition, inference throughput is around 555 img/s which shows efficient execution of the model. When faults are introduced, throughput experiences a moderate drop of around 93.7% to 94.9% in comparison with the baseline, which indicates faults alone won't reduce processing speed; the trade-off is known only when

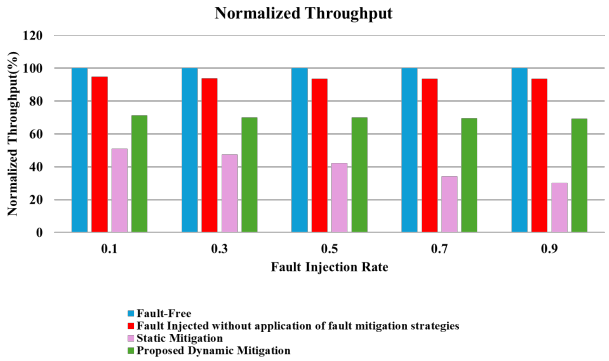


Figure 8 – Impact of fault injection and mitigation on inference throughput

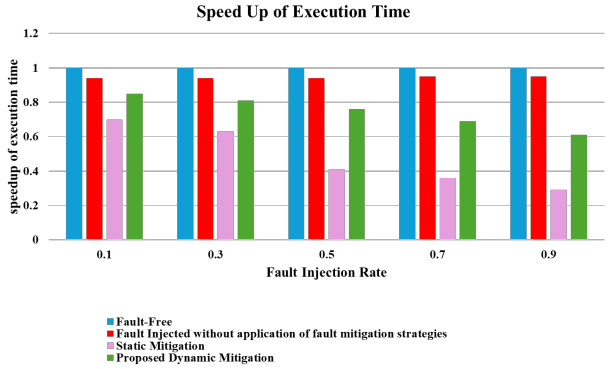


Figure 9 – impact of fault injection and mitigation on inference execution time

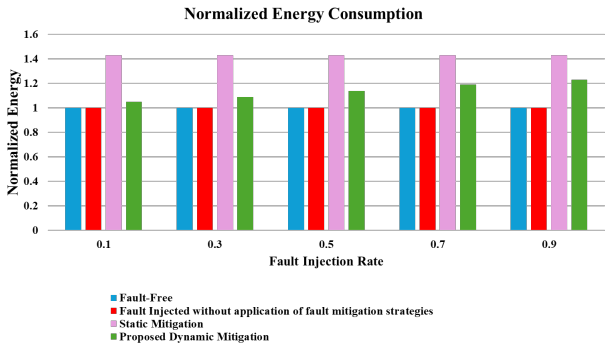


Figure 10 – Impact of fault injection and mitigation on energy consumption during inference

mitigation strategies applied. Static mitigation results in a continuous decline in throughput as fault injection rate increases which indicates that the constant recalculation for all fault severity levels reduces the processing speed. But dynamic mitigation maintained a stable throughput by dynamic application of fault mitigation strategies, confirming that it balances the reliability and efficiency of the model.

5.4 Inference execution time analysis

The graph below in Fig. 9 presents the inference execution time. The x-axis represents the fault injection rate, while the y-axis indicates x times speedup of execution time.

The baseline model executes in 18.03 seconds, demonstrating efficient execution of the model under normal conditions. Once the faults are injected, the model's execution time slightly increases as the fault rate increases due to the minor stalls in processing caused by faults. Static mitigation that applied the mitigation strategy inefficiently exhibited heavy overhead and slows down execution, especially at higher fault rates. In contrast, dynamic mitigation strategies maintain higher execution values, ranging from 0.85× to 0.61× times in comparison with the baseline model across all fault injection rates. This stability emphasizes the efficiency of dynamic mitigation strategies, where only critical faults are addressed, mainly reducing unnecessary recalculations. This enables the model to maintain faster execution in comparison with the static mitigation strategy.

5.5 Inference energy consumption analysis

The graph Fig. 10 presents the inference energy consumption. The x-axis represents the fault injection rate, while the y-axis measures normalized energy consumption.

In both the baseline and faulty conditions, energy consumption remains low and stable. This indicates that a faulty model alone does not essentially impact power usage. Still, when mitigation strategies are applied, energy consumption increases significantly due to additional operations required for fault detection and mitigation.

Static mitigation shows that the energy consumption increases more as fault rate increases. This reflects the need for extra computational overhead from consistently applying a fault mitigation strategy for every potential fault. Regardless of the fault severity level, each fault is recalculated to improve the model's performance in terms of accuracy and loss. In contrast, dynamic mitigation achieves a more energy-efficient method by selectively applying fault mitigation strategies. Energy consumption under dynamic mitigation scales more moderately, increasing from 3.05× to 7.22×, demonstrating a better balance between performance recovery and usage of system resources.

Overall, dynamic mitigation significantly outperforms static correction in maintaining energy efficiency under increasing fault severity, validating its deployment in real-time ViT hardware accelerators.

6. CONCLUSION

This paper offers a fault-tolerant Vision Transformer (ViT) architecture designed to improve resilience against

permanent hardware faults while preserving computational efficiency. Using a centralized BIST unit and a learning-based decision algorithm, the proposed architecture combines real-time fault detection and adaptive mitigation strategies based on fault severity classification. The system dynamically decides whether calculations should be recomputed using a lightweight recompute unit or assigned to healthy PEs, or bypassed by means of methodical monitoring and classification.

Experimental results validate the effectiveness of this method. The fault-tolerant ViT model achieves 77.68% accuracy at 0.9 fault injection rate, while baseline accuracy is 80.8% which demonstrates the robustness of the proposed system. The fault injection causes a sharp increase in loss, which is similar to trends observed in accuracy analysis that is further reduced through mitigation mechanisms. The other performance metrics of the system such as throughput, execution time, and energy consumption metrics similarly reflect the trade-offs, indicating modest throughput degradation, slight increases in execution time, and marginal energy overhead under various fault conditions, which are altogether effectively managed by the mitigation framework. These findings show that the proposed decision-driven fault-tolerant ViT architecture effectively mitigates hardware-induced errors while preserving accuracy and performance with minimal computational overhead.

REFERENCES

[1] A. Dosovitskiy et al. *An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale*. ArXiv.org, Last retrieved 2021-06-03. June 2021.

[2] S. Nag, G. Datta, S. Kundu, N. Chandrachoodan, and P. A. Beerel. "ViTA: A Vision Transformer Inference Accelerator for Edge Applications". In: *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*. Monterey, CA, USA, 2023, pp. 1–5. DOI: 10.1109/ISCAS46773.2023.10181988.

[3] Kai Han, Yunhe Wang, An Xu, Chunjing Chen, Jianyuan Guo, Chang Xu, Chao Xu, and Dacheng Tao. "A Survey on Vision Transformer". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.1 (Jan. 2023), pp. 87–110. DOI: 10.1109/TPAMI.2022.3152247.

[4] L. Papa, P. Russo, I. Amerini, and L. Zhou. "A Survey on Efficient Vision Transformers: Algorithms, Techniques, and Performance Benchmarking". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46.12 (Dec. 2024), pp. 7682–7700. DOI: 10.1109/TPAMI.2024.3392941.

[5] N. Fnu and A. Bansal. "Understanding the Architecture of Vision Transformer and Its Variants: A Review". In: *2024 1st International Conference on Innovative Engineering Sciences and Technological Research (ICIESTR)*. Muscat, Oman, 2024, pp. 1–6. DOI: 10.1109/ICIESTR60916.2024.10798341.

[6] J. Maurício, I. Domingues, and J. Bernardino. "Comparing Vision Transformers and Convolutional Neural Networks for Image Classification: A Literature Review". In: *Applied Sciences* 13.9 (2023), p. 5521. DOI: 10.3390/app13095521. URL: https://doi.org/10.3390/app13095521.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010. ISBN: 9781510860964.

[8] JunSeok Lim and KoengMo Sung. "FNN (Feedforward Neural Network) Training Method Based on Robust Recursive Least Square Method". In: *Advances in Neural Networks – ISNN 2007*. Ed. by Derong Liu, Shumin Fei, Zengguang Hou, Huaguang Zhang, and Changyin Sun. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 398–405.

[9] Y. Zhao, K. Wang, and A. Louri. "FSA: An Efficient Fault-tolerant Systolic Array-based DNN Accelerator Architecture". In: *2022 IEEE 40th International Conference on Computer Design (ICCD)*. Olympic Valley, CA, USA, 2022, pp. 545–552. DOI: 10.1109/ICCD56317.2022.00086.

[10] H. Chen, X. Zhang, K. Huang, and F. Koushanfar. "AdaTest: Reinforcement Learning and Adaptive Sampling for On-chip Hardware Trojan Detection". In: *ACM Transactions on Embedded Computing Systems* 22.2 (Mar. 2023), Article 37.

[11] T. Wang, Y. Zhang, J. Wang, J. Liu, and Z. Lu. "ViA: A Novel Vision-Transformer Accelerator Based on FPGA". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41.11 (Nov. 2022), pp. 4088–4099. DOI: 10.1109/TCAD.2022.3197489.

[12] Ke Wang, Hao Zheng, Yuan Li, Jiajun Li, and Ahmed Louri. "AGAPE: Anomaly Detection with Generative Adversarial Network for Improved Performance, Energy, and Security in Many-core Systems". In: *Proceedings of Design, Automation Test in Europe Conference Exhibition (DATE)*. Antwerp, Belgium, 2022.

[13] Ke Wang, Ahmed Louri, Avinash Karanth, and Razvan Bunescu. "IntelliNoC: A Holistic Design Framework for Energy-Efficient and Reliable On-Chip Communication for Manycores". In: *Proceedings of the 46th Annual International Symposium on Computer Architecture (ISCA)*. Phoenix, AZ, USA, 2019, pp. 1–12. DOI: 10.1145/3307650.3322263.

[14] K. Wang, H. Zheng, J. Li, and A. Louri. "Morph-GCNX: A Universal Architecture for High-Performance and Energy-Efficient Graph Convolutional Network Acceleration". In: *IEEE Transactions on Sustainable Computing* 9.2 (Mar. 2024), pp. 115–127. DOI: 10.1109/TSUSC.2023.3313880.

[15] X. Xue et al. *ApproxABFT: Approximate Algorithm-Based Fault Tolerance for Vision Transformers*. ArXiv (Cornell University), Last retrieved 2023-01. Jan. 2023.

[16] Alama et al. *Algorithmic Strategies for Sustainable Reuse of Neural Network Accelerators with Permanent Faults*. ArXiv (Cornell University), Last retrieved 2024-12. Dec. 2024.

[17] H. Jian et al. "PerFT-N: Low-Overhead Permanent Fault-Tolerance Mechanism for Neural Processing Units". In: *Proceedings of the Great Lakes Symposium on VLSI 2022*. June 2024.

[18] 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: *2018 IEEE 36th VLSI Test Symposium (VTS)*. San Francisco, CA, USA, 2018, pp. 1–6.

[19] F. Fernandes dos Santos et al. "Improving Deep Neural Network Reliability via Transient-Fault-Aware Design and Training". In: *IEEE Transactions on Emerging Topics in Computing* (2024). DOI: 10.1109/TETC.2024.3520672.

[20] U. Zahid et al. *FAT: Training Neural Networks for Reliable Inference under Hardware Faults*. ArXiv.org, Last retrieved 2020-11-11. Nov. 2020.

[21] Seo-Seok Lee and Joon-Sung Yang. "Value-aware Parity Insertion ECC for Fault-tolerant Deep Neural Network". In: *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2022, pp. 724–729. DOI: 10.23919/DATE54114.2022.9774543.

[22] Tooba Arifeen et al. "A Fault Tolerant Voter for Approximate Triple Modular Redundancy". In: *Electronics* 8.3 (2019), pp. 332–332.

[23] Y. Wu and S. Adham. "BIST fault diagnosis in scan-based VLSI environments". In: *Proceedings International Test Conference 1996: Test and Design Validity*. Washington, DC, USA, 1996, pp. 48–57. DOI: 10.1109/TEST.1996.556944.

[24] F. Meng. "Self-Testing and Self-Healing Neural Network Accelerator Design". Order No. 30250187. PhD thesis. University of Delaware, 2023.

[25] M. A. Hanif and M. Shafique. "FAQ: Mitigating the Impact of Faults in the Weight Memory of DNN Accelerators through Fault-Aware Quantization". In: *2023 International Joint Conference on Neural Networks (IJCNN)*. Gold Coast, Australia, 2023, pp. 1–8.

[26] Yingnan Zhao, Ke Wang, and Ahmed Louri. "FSA: An Efficient Fault-tolerant Systolic Array-based DNN Accelerator Architecture". In: *2022 IEEE 40th International Conference on Computer Design (ICCD)*. 2022, pp. 545–552. DOI: 10.1109/ICCD56317.2022.00086.

[27] Joonsang Yu, Junki Park, Seongmin Park, Minsoo Kim, Sihwa Lee, Dong Hyun Lee, and Jungwook Choi. "NN-LUT: neural approximation of non-linear operations for efficient transformer inference". In: *Proceedings of the 59th ACM/IEEE Design Automation Conference*. DAC '22. San Francisco, California: Association for Computing Machinery, 2022, pp. 577–582. ISBN: 9781450391429. DOI: 10.1145/3489517.3530505. URL: https://doi.org/10.1145/3489517.3530505.

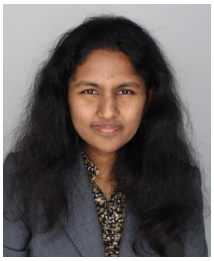
[28] Shin-Yeu Lin. "Basic hardware module for a nonlinear programming algorithm and applications". In: *Automatica* 33.8 (1997), pp. 1579–1586. ISSN: 0005-1098. DOI: https://doi.org/10.1016/S0005-1098(97)00053-8. URL: https://www.sciencedirect.com/science/article/pii/S0005109897000538.

[29] Ke Wang and Ahmed Louri. "CURE: A High-Performance, Low-Power, and Reliable Network-on-Chip Design Using Reinforcement Learning". In: *IEEE Transactions on Parallel and Distributed Systems* 31.9 (2020), pp. 2125–2138. DOI: 10.1109/TPDS.2020.2981284.

[30] Ke Wang, Hao Zheng, and Ahmed Louri. "Systems and methods for learning-based high-performance, energy-efficient, and secure on-chip communication design framework". US12238126B2. Feb. 2025. URL: https://patents.google.com/patent/US12238126B2/en.

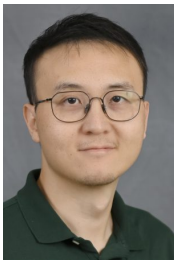
[31] D. DiTomaso, T. Boraten, A. Kodi, and A. Louri. "Dynamic error mitigation in NoCs using intelligent prediction techniques". In: *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Taipei, Taiwan, Province of China, 2016, pp. 1–12.

AUTHORS



Indhuja Gudluru received Master's degree in computer engineering from the University of North Carolina at Charlotte in 2025 and BE in electronics and communication engineering from Anna University in 2021. Her research interests include machine learning and deep learning, AI hardware accelerators, and networking. She is passionate about artificial intelligence and its potential to drive innovation across systems and architectures.

Chunyuan Shen received his BS degree in software engineering from the Beijing Jiaotong University, China in 2022. He is currently working toward a Ph.D. degree in electrical engineering at the University of North Carolina at Charlotte. His research interests include vision transformer, Al accelerator design, computer architecture, and interconnection networks.



Ke Wang received his Ph.D. degree in computer engineering from the George Washington University in 2022. He received M.S. degree in electrical engineering from Worcester Polytechnic Institute in 2015, and B.S. degree in Electrical Engineering from Peking University in 2013. He is currently an assistant professor of electrical and computer engineering at the University of North Carolina at Charlotte. His research work focuses on parallel computing, computer architecture, interconnection networks, and machine learning.