Auto-Healer: Self-Healing Hardware for Perception Stage Faults in Autonomous Driving Systems

Ali Suvizi
George Washington University
Washington DC, DC, USA
ali.suvizi@gwu.edu

Guru Venkataramani
George Washington University
Washington DC, DC, USA
guruv@gwu.edu

Abstract

Autonomous Driving Systems (ADS) are safety-critical applications designed to enhance vehicle autonomy and general road safety. Perception, a critical stage of the pipeline, detects and classifies objects and environmental conditions. This stage is also highly susceptible to faults such as transient and permanent bit flips, which compromise the entire ADS pipeline. To address such reliability challenges, our paper proposes *Auto-Healer*, a *novel* self-healing hardware architecture customized for the perception stage to enhance the reliability and fault tolerance of ADS. We propose a chiplet-based architecture with an FPGA that performs automated fault detection in ADS and error correction with negligible latency ($\sim$ 0.08%) and power overhead of ($\sim$ 0.8 W) compared to a system without self-healing support. We leverage Dual Modular Redundancy (DMR), typically integrated into many modern ADS systems, for fault management in our design. *Auto-Healer* dynamically adapts to faults by efficiently accelerating image processing for seamless recovery, enabled by the flexibility and reconfigurability of FPGA technology. The self-healing system demonstrates low Mean Time to Repair (MTTR), measured at 40 ns for transient faults and 120 ns for permanent faults, resulting in negligible latency overheads (less than 0.001%) compared to the system with no self-healing support.

CCS Concepts

• **Hardware** → **Reconfigurable logic and FPGAs**; **Safety critical systems**; • **Computer systems organization** → **Reliability**.

ICS '25, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-1537-2/25/06
https://doi.org/10.1145/3721145.3725755

Keywords

Autonomous Driving Systems, Self-Healing Hardware, Deep Neural Networks, Field Programmable Gate Arrays

ACM Reference Format:
Ali Suvizi and Guru Venkataramani. 2025. Auto-Healer: Self-Healing Hardware for Perception Stage Faults in Autonomous Driving Systems. In *2025 International Conference on Supercomputing (ICS '25), June 08–11, 2025, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3721145.3725755

1 Introduction

Autonomous Driving Systems (ADS) (comprising Advanced Driving Assistance Systems and Automatic Vehicles), constitute a paradigm shift towards making automotive vehicles safer, alleviating traffic congestion, and unfolding new dimensions for user mobility solutions. Leading industry players like Google [50], Tesla [58], and NVIDIA [34] have made huge strides in this domain [3, 36]. The ADAS/AV systems can respond quicker to dynamic road conditions than human drivers by utilizing advanced technologies for object detection, localization, and decision-making, hence improving their overall safety [63]. Despite many of these huge benefits, the reliability of such systems remains one of the most salient issues in the deployment of ADS.

In this work, we focus on improving the reliability of ADS, especially in their perception stage, which is crucial for real-time decision-making and is one of the most vulnerable stages in the ADS pipeline [15]. The perception stage takes the raw sensor data in the form of images, cameras, LiDAR, and radar and deduces from this information object detection, lane detection, road condition analysis, and an understanding of the driving environment [20]. A single mistake or delay in this critical step may lead to catastrophic consequences [13]; for example, missing object detections or incorrect obstacle avoidance decisions may make the complete system unsafe and nonfunctional [15, 60].

Convolutional neural networks (CNNs) are widely adopted at the perception stage for image classification and object detection. The multi-layer nature of CNNs has shown them to be subject to transient and permanent faults [15, 57]. Fully connected (FC) layers are further vulnerable to errors as these tend to aggregate information from all previous layers.

In the latter layers, defects can directly affect the output, causing misclassification that is particularly dangerous to safety-critical ADS tasks [21, 39]. This situation is further exacerbated by permanent faults [23, 47], for instance, stuck-at faults owing to time-dependent dielectric breakdown (TDDB) and electromigration, among other aging mechanisms.

Traditional fault tolerance mechanisms, like Triple Modular Redundancy (TMR) and Error Correcting Code (ECC), have significant deficiencies in meeting the timing demands of ADS. TMR masks the faults by redundancy but does not correct errors; it has no end-to-end fault management, hence making the system susceptible to persistent or cascading failures [49]. ECC works well for transient faults but faces difficulties with fault scenarios that are more complex and with a wider failure spectrum. Also, both of the approaches have significant latency, energy overheads, and costs [29, 58], which contradicts the real-time processing, energy efficiency, and cost-effectiveness required by ADS pipelines [66].

To tackle the above challenges, we introduce *Auto-Healer*, a novel chiplet-based self-healing hardware architecture tailored to address the vulnerabilities of the perception stage in the ADS pipeline, particularly in image processing and object detection. The proposed mechanism ensures robust CNN operations with minimal performance overheads, even under stringent real-time constraints, by autonomously detecting, diagnosing, and recovering from faults. Leveraging FPGA-based technology, our proposed architecture combines high performance, scalability, and adaptability, meeting ADS systems' reliability and fault-tolerance demands. Through our proposed design, we aim to significantly enhance the safety and dependability of autonomous driving technologies while ensuring minimal overhead on latency and energy consumption. This enables ADS to perform rapid decision-making, ensuring timely responses before critical deadlines, thus paving the way for resilient and efficient automotive systems in the future.

In summary, the key contributions of our paper are:

- We propose *Auto-Healer*, a **novel self-healing architecture** to address the perception-stage bit faults within the ADS pipeline that does image processing and object detection. To the best of our knowledge, this is the *first self-healing mechanism in the ADS domain* that seeks to integrate autonomous fault detection, diagnosis, and self-healing to provide robustness and reliability for critical real-time ADS operations.
- We design *Auto-Healer* using **FPGAs** that combine three key benefits: high performance through hardware acceleration, energy efficiency achieved by low resource usage, and adaptability to faults and changing conditions (supporting reconfigurability for dynamic fault recovery).
- We explore the benefits of **active-passive fault management with DMR** for improved fault detection and recovery. Our architecture employs lightweight synchronization points, which trigger the self-healing mechanism only in cases of disagreement between the two modules, thus allowing for improved energy efficiency and operational effectiveness.
- Our experimental evaluation demonstrates **minimal performance overheads** for the self-healing mechanism, with a latency increase of only 40 ns for transient faults and 120 ns for permanent faults compared to a baseline with no healing functions. The system achieves a total execution time of 0.95 ms for the *CNN without self-healing* and just 92 ns more *with self-healing*, ensuring minimal disruption. The exceptionally low Mean Time to Repair (MTTR) of 40 ns for transient faults and 120 ns for permanent faults further highlights the system's reliability and timeliness. These results demonstrate robust performance and timely decision-making, satisfying the stringent requirements of ADS applications.

This paper is structured as follows: Section 2 provides background on autonomous driving, ADS safety standards, and self-healing mechanisms. Section 3 outlines the threat model, highlighting perception-stage vulnerabilities to bit flip faults. Section 4 discusses the limitations of the current design. Section 5 details the self-healing architecture and FPGA features. Section 6 describes the implementation. Section 7 presents experimental results on MTTR, latency, and power consumption. Section 8 reviews related fault tolerance work and Section 9 concludes with insights and future directions.

2 Background

2.1 Autonomous Driving

To facilitate the advancement of AVs, the National Highway Traffic Safety Administration (NHTSA) established guidelines referencing the six levels of automation defined by the Society of Automotive Engineers (SAE) International [48, 64]. These levels range from No Automation (Level 0), where the driver retains complete control, to Full Automation (Level 5), where the automated system assumes full driving responsibility under all conditions.

The functions assigned to an actual autonomous vehicle, that is, ADAS or ADS, are primarily divided into a pipeline processing system comprising three primary tiers as shown by [37]: *Perception*, *Planning*, and *Control*. The perception module processes input from sensors such as cameras, LiDAR, GPS, IMU, and radar by executing key tasks, including localization, object classification, object detection, and object

Table 1: Comparison of Automation Levels, Platforms, and Sensors Across Leading Manufacturers

Manufacturer	Mobileye[38]	Tesla[58]	Audi[4, 34]	Waymo[50]
Automation	Level 2	Level 2	Level 3	Level 4
Platform	SoCs	SoCs	SoCs	SoCs
Platform Description	SuperVision System	Full Self-Driving	AI-Based Platform	Waymo Proprietary Suite
Sensor	Camera	Camera, Radar	Lidar, Camera, Radar	Lidar, Camera, Radar

tracking, allowing the vehicle to gain a detailed understanding of its immediate environment. Mission planning resolves the definition of the best path the vehicle will decide from the current position by using some algorithms like A* [22]. The vehicle's motion is calculated through motion planning, and the steering, acceleration, and braking are governed by dynamic movement primitives [32]. A control module will execute the trajectory from the motion planning module and translate it to low-level commands for actuators with great accuracy. Using techniques such as PID controllers [62] or Model Predictive Control (MPC) [51], the controller ensures proper steering, acceleration, and braking and continuously adjusts itself to maintain the safety and stability of the vehicle while operating. A detailed diagram of ADS components is presented in Figure 1.

2.2 ADS Functional Safety Standards

In the automotive industry, the functional safety requirements are categorized into Automotive Safety Integrity Levels (ASILs) ranging from ASIL-A to ASIL-D, with ASIL-D representing the highest safety risk [18]. Higher ASIL levels, especially ASIL-D, demand aggressive safety features, including sophisticated error detection and recovery mechanisms such as lockstep execution, which are essential for ASIL-D micro-controller units (MCUs).

ISO26262 enforces independent redundancy at higher levels of ASIL to control the occurrence of common-cause failures. Staggered DMR, in which one core runs a fixed execution cycle behind the other core so as to enable the detection of identical errors [61], is widely used in automotive systems. Redundancy can be applied at several levels depending on the Sphere of Replication (SoR), which impacts both system cost and fault detection latency [46].

2.3 Self-Healing

Self-healing systems represent a paradigm shift in designing fault-tolerant and resilient architectures, particularly for safety-critical applications like ADS. These systems are engineered to autonomously detect, diagnose, and repair faults during runtime to ensure system availability, reliability, and safety. This capability is particularly vital in ADS, where faults in perception or decision-making can result in catastrophic consequences.

2.3.1 Definition and Key Components. A self-healing system is built upon an adaptive feedback loop often modeled after the Observe-Orient-Decide-Act (OODA) framework [7]. This loop integrates several interdependent components. Fault detection mechanisms monitor the system for anomalies, such as transient and permanent bit flips, synchronization errors, or performance degradation. Once anomalies are identified, fault diagnosis processes isolate and identify their root causes, whether they arise from hardware malfunctions or computational software errors. Recovery mechanisms then implement corrective actions, such as checkpointing, rollback, or input filtering, to ensure system continuity. Finally, testing and adaptation mechanisms validate recovery measures and dynamically adjust to evolving fault conditions using runtime monitoring (Fig. 2).

2.3.2 Need for Self-Healing Architectures in ADS. ADS are integral to modern vehicles, enhancing safety and driving comfort through features such as adaptive cruise control, lane-keeping assistance, and emergency braking systems. These systems rely heavily on CNNs for tasks such as object detection and image classification, which are computationally intensive and susceptible to transient and permanent errors like bit flips [24, 28]. Such errors can easily propagate through the network layers, leading to large inaccuracies compromising safety-critical decisions [12, 67].

Functional safety standards, such as ISO 26262 and ASIL-D, require stringent fault tolerance for the highest benchmarks of safety and reliability. Self-healing systems are vital for safer and more reliable autonomous driving technologies, as they address vulnerabilities inherent in CNN-based perception tasks and operate under real-time constraints [2, 16, 17].

3 Threat Model

3.1 Faults in ADS Pipeline Stages

The perception stage is one of the most sensitive in the ADS pipeline chain, as it receives a large volume of raw sensor data for object detection and classification and is highly vulnerable to faults due to the dynamic environment of autonomous driving [14, 35]. Table 2 provides the frequency of bugs in various ADS components, where perception-related tasks and object detection represent a significant population. The data extracted from Apollo [5] and Autoware [59] benchmarks in [15] reveals that object detection accounts for a large fraction of bugs-55 instances, making it one of the most fault-prone sub-components in the ADS pipeline.

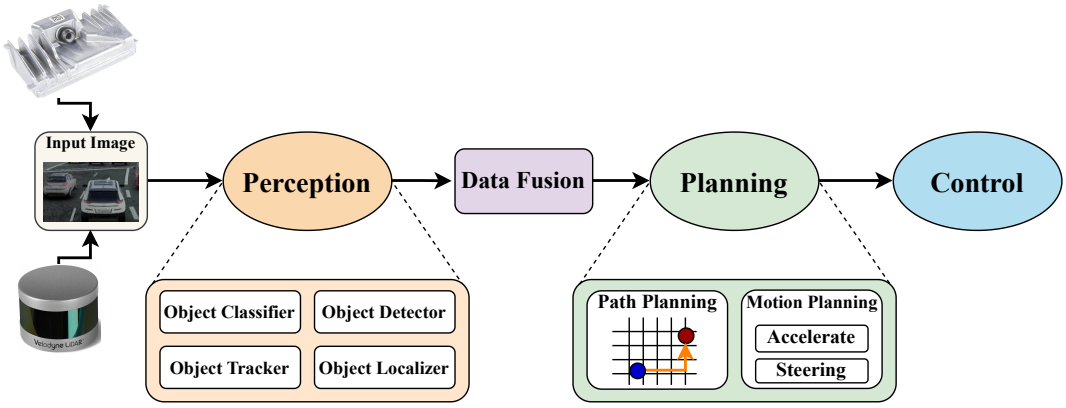

Figure 1: An illustration of the Autonomous Driving Systems Pipeline

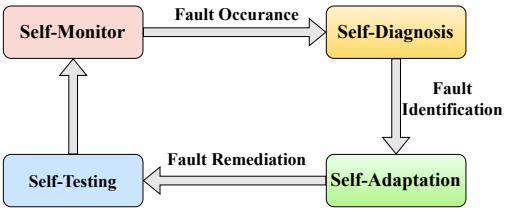

Figure 2: Self-Healing Loop Architecture

Table 2: Frequency of bug occurrences for each AV component, highlighting the vulnerability of perception tasks, including detection and localization, in ADS pipelines.

Component	Sub-Component	Apollo	Autoware	Total
Perception	**Object Detection**	17	38	**55**
	Object Tracking	2	9	11
	Data Fusion	11	6	17
	Multi-Sensor Fusion	9	21	30
	Lidar Locator	1	26	27
Planning	Path Planning (Map)	13	5	18
	Trajectory Prediction	7	1	8
Control	Control	4	0	4

3.2 Fault Types

The perception stage is inherently vulnerable to faults due to its reliance on CNNs for image classification, object detection, and environmental mapping tasks. Two primary types of faults significantly impact this stage [24]:

Transient Faults: Transient faults, such as bit flips [8, 42], usually arise from environmental factors such as cosmic radiation, electromagnetic interference, and thermal fluctuations [45]. These faults can easily propagate over the network, scaling up inaccuracies and undermining safety-critical decisions, such as failing to detect an obstacle or misclassifying road conditions [27]. Transient faults manifest in various forms. Due to environmental disturbances, single-bit upsets involve the inversion of a single memory or register bit. In contrast, multiple-bit upsets occur when several bits are simultaneously flipped, increasing the complexity of error correction.

Permanent Faults: Permanent faults, including stuck-at faults and permanent bit flips [1, 8], generally arise due to hardware aging mechanisms like time-dependent dielectric breakdown (TDDB) and electromigration. These faults cause permanent malfunctions in components like multipliers and accumulators, as well as persistent corruption of stored data, degrading the system's overall performance over time [14, 23]. Such faults badly affect real-time decision-making in the perception stage because faulty hardware and corrupted data may lead to persistent errors in computations, further compromising the integrity of the ADS pipeline [28, 42]. One example of permanent faults includes single hard errors, which result in logic states being stuck at zero or one, leading to persistent failures in memory or registers [11, 45]. A detailed categorization of fault types can also be found in Figure 3.

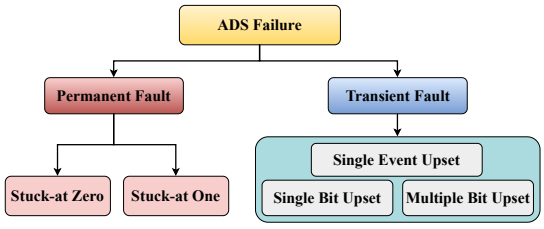

Figure 3: Fault types considered by our work

4 Limitations of Current Designs

Modern ADS increasingly rely on computationally intensive tasks such as object detection and classification, which demand high levels of reliability and real-time performance. However, these systems are bound to significant challenges that compromise their operational efficiencies and safety: (1) stringent latency requirements make every slightest delay unacceptable [19, 36]; (2) transient/permanent faults in computational units may cause catastrophic failures [1, 24]; and (3) legacy fault-tolerance mechanisms are frequently costly [58] and introduce and overheads that violate with the real-time constraints of the ADS pipelines [2].

Conventional TMR and ECC solutions have well-known limitations in satisfying strict ADS requirements. TMR primarily masks the faults by redundancy but does not include error correction or end-to-end fault management capabilities, thereby exposing the systems to persistent or cascading failures [49]. Similarly, ECC works well for transient faults like one or two bit flips but fails to handle multi-bit error scenarios. Both of the methods incur high latency, energy overheads, and costs [10, 58].

Given such challenges, we propose a chiplet-based self-healing architecture in an ADS system involving FPGAs. The proposed architecture of automated fault detection, diagnosis, and correction will leverage the flexibility and reconfigurability of FPGAs with an assurance of high reliability in fault-inducing scenarios.

5 Design Overview of Auto-Healer

5.1 Key Features

The *Auto-Healer* self-healing architecture integrates fault tolerance and real-time processing capabilities specifically tailored for ADS. Its key features include:

- **Focus on MAC Operations**: Prioritizes fault detection and correction for Multiply-Accumulate (MAC) operations, making it applicable to any computational system heavily relying on MAC operations.
- **Generic and Flexible Design**: Incorporates dynamic synchronization point configurations, enabling adjustable fault comparison frequency and checkpoint counts. Supports fault tolerance across all ADS pipeline stages.
- **Dual-Level Parallelism**: Leverages parallelism within and across CNN layers, reducing latency and enhancing efficiency while ensuring real-time processing.
- **Resource Efficiency and Scalability**: Stores weights and biases in BRAM to minimize DSP, LUT, and FF usage. Allows deployment of up to four CNN instances on a single FPGA, with two active and two passive modules ensuring fault tolerance.
- **Advantages Over TMR and ECC/CRC**: Unlike TMR, which incurs significant power and latency overhead by running all three modules simultaneously and only masks faults through majority voting, *Auto-Healer* detects, corrects, and resolves faults efficiently. It dynamically isolates and repairs permanent faults while ensuring minimal operational overhead. In contrast to ECC and CRC, *Auto-Healer* avoids complex encoding/decoding logic, providing direct and efficient correction for both data and logic faults, ensuring long-term system reliability.
- **Checkpointing for Fault Recovery**: Stores checkpoints in LUTRAM for fast recovery and efficient fault resolution, ensuring minimal disruption to ADS operations under stringent latency and availability constraints.

5.2 Criteria for Self-Healing System Design

In designing self-healing architectures for ADS, much emphasis is placed on continuous availability and survivability under fault-inducing scenarios. As illustrated in Fig. 4, these systems automatically detect and diagnose faults, recover, and isolate them using reconfiguration and synchronization techniques that enhance fault tolerance with minimal disruption. This automated fault management assures real-time adaptability and reliability; hence, it is indispensable in safety-critical applications like ADS.

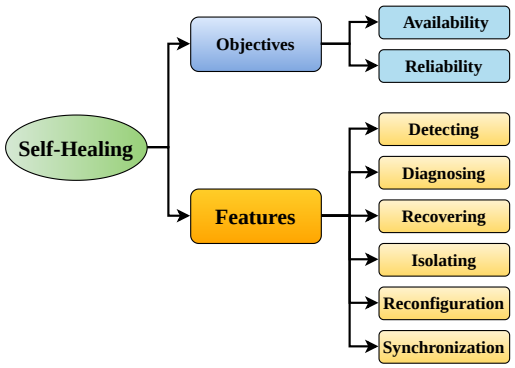

Figure 4: Self-Healing Properties

5.3 Auto-Healer Modules

Our proposed self-healing architecture, as shown in Fig. 5, integrates self-fault detection, self-diagnosis, and self-correction mechanisms into a unified framework, leveraging FPGA-based flexibility and reconfigurability. The architecture is designed to address both transient and permanent faults while ensuring minimal disruption to ADS operations. Below, we describe its key components and functionality.

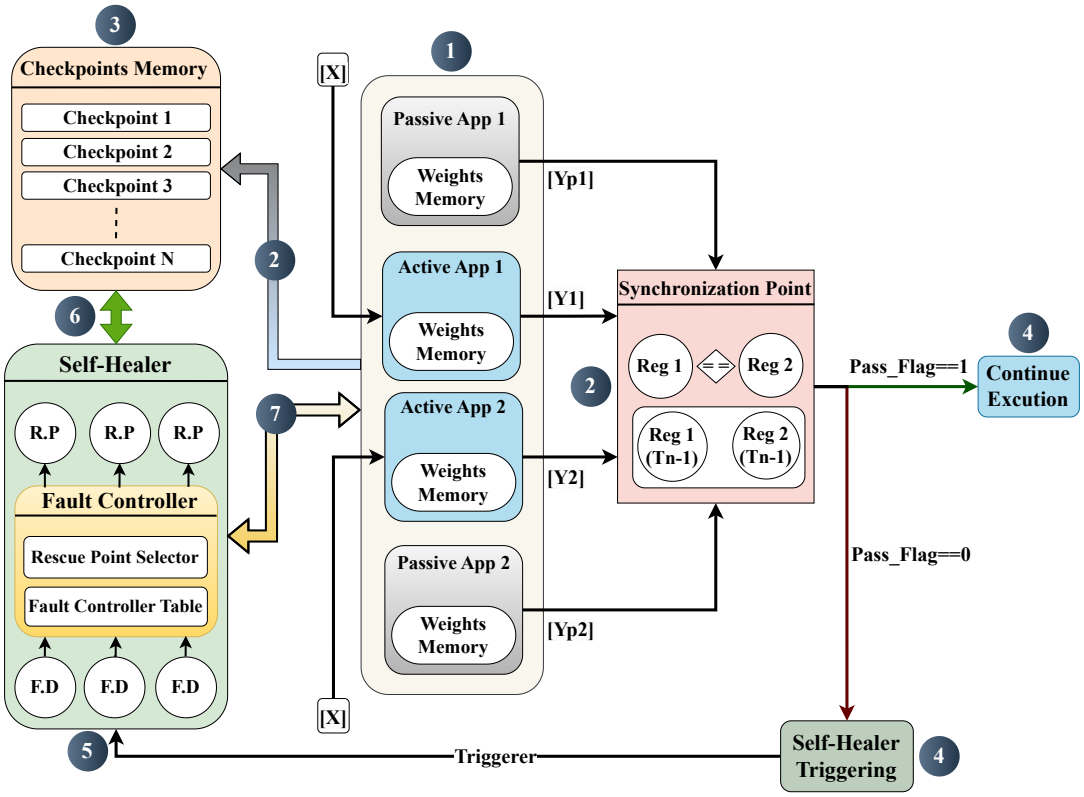

Figure 5: Self-Healing Architecture

5.3.1 ***Active-Passive Module Design.*** The architecture is centered around the active-passive module pair using DMR ❶. An equivalent passive module can replicate every computation performed by an active module. At any given time, the active modules share the same processing unit for CNN computations and take in the same input image. Local weights and biases are used for the computations, which are stored within each processing unit. Each of these pairs executes the image processing job concurrently, sending results to the synchronization point for checking. The results are sent at every MAC operation to this synchronization point to have a fine granular level of fault detection in this design ❷. During normal execution, checkpoints are saved simultaneously as data is sent to the synchronization point ❷. If DMR module outputs do not match, the most recent checkpoint is discarded as it is considered faulty, and the last stable checkpoint is retained. The passive modules add redundancy for robust fault tolerance, increasing system reliability considerably, while the whole process of fault detection, triggering, and fallback to passive modules is done autonomously.

5.3.2 ***Synchronization Point.*** Synchronization point ❷ is crucial for comparing active modules' outputs. This is the point at which the system will either act in a fault-free state or there is some discrepancy. The synchronization point stores interim values that might be useful for diagnosing faults in case of recovery, achieved through adding temporary registers. When there is a match in outputs, the system will work fine and smoothly, considering the last saved checkpoint as an indicator ❹. In contrast, this same synchronization point would invoke the fault detection mechanism if something is wrong ❹. Due to the generic and adaptive nature of their design, the frequency of comparisons may be adjusted dynamically. This flexibility ensures that synchronization point monitor the system effectively while maintaining operational integrity.

5.3.3 ***Fault Detection Mechanism.*** The fault detection mechanism ❺ classifies discrepancies as transient or permanent faults and their nature. Triggered by a synchronization point ❹, fault detectors (F.D) assigned to different parts of the system, for example, CNN layers or specific computational stages, initiate the re-execution of the computational step from the last stable checkpoint ❻, ❼. This re-execution helps to classify the fault into transient-for example, bit flips due to external interference-or permanent-for instance,

stuck-at faults. Transient faults are cleared after the first re-execution and are verified by synchronized outputs. If the discrepancy remains, then detectors identify the type and location of the fault, using checkpoint data to determine the exact application, CNN layer, and position of the computed data responsible for the fault. It is fully automated, meaning fault diagnosis will be timely and accurate without human intervention.

5.3.4 ***Fault Controller and Rescue Points.*** A fault controller co-operates with the rescue points (R.P) in order to carry out the process of fault recovery ❺. In case of confirmation of permanent fault, the fault controller would investigate the application, CNN layer, and the position of occurrence using knowledge from detectors and checkpoints ❻. This information is kept in the fault controller table (F.C.T) for future reference. In case of a transient fault, the fault controller enables simple re-execution through rescue points, where the problem does not cause the firing of the passive modules. In case of a permanent fault, the fault controller isolates the faulty application and switches on the passive module with the last stable checkpoint to keep the system running ❼. Then, the self-healing mechanism identifies the faulty application(s) with the help of information from the synchronization point ❷ by comparing the result of new active applications with the old ones and selects the best rescue point for its recovery based on the fault type and its location. The faulty application gets updated with the corrected data and now turns into the new passive module. Thus, all these processes happen automatically for a high degree of system reliability with minimal disturbance to the operations.

5.3.5 ***Fault Recovery and Checkpointing.*** Fault recovery leverages rescue points to replay computation steps for recovering faulty data. The checkpointing mechanism stores full snapshots of system states that capture critical operational information, including the type of operation (multiplication or addition), CNN layer number, data location within matrices, and the result values of addition and multiplication operations ❸. These checkpoints are recorded after the synchronization point verifies that no discrepancy exists. If the comparison at the synchronization point reveals no mismatch, the system state is saved as a checkpoint for potential future recovery ❷ by comparing the result of new active applications with the old. The checkpointing system is implemented in hardware, specifically designed for FPGA. To optimize performance, storage is allocated to LUTRAM instead of BRAM. This choice leverages the faster read/write capabilities of LUTRAM and its abundance on FPGA platforms compared to the relatively limited availability of BRAM. Additionally, checkpoints are managed using a First-In-First-Out (FIFO) strategy to optimise memory usage, maintaining only the last three snapshots. This generic design allows the system to adjust the number of stored checkpoints as needed, but for the current application, three checkpoints suffice to address transient bit flips and stuck-at faults. In a fault-free scenario, the system efficiently processes data without the need to access the checkpoints, ensuring minimal latency. When faults occur, the most recent checkpoint is discarded, and the remaining two are used to recover faulty computations. This approach not only saves memory and reduces latency but also ensures the highest level of reliability under all conditions, meeting stringent ADS requirements for execution time and availability, even during fault recovery.

When a fault occurs, the last stable checkpoint contains the output of the last correctly executed MAC operation. This output serves as the input for the MAC operation where the fault occurred. The fault controller determines the optimal R.P based on the fault's location and type. The selected R.P retrieves the necessary data ❻ and supplies it to the application, enabling seamless re-execution of the faulty operation ❼. During recovery, transient faults can be resolved by re-executing the operation using the last stable checkpoint. For permanent faults, the system isolates the faulty module, activates a passive module, and continues execution while correcting the faulty application using rescue points. The structured workflow of the self-healing architecture for ADS perception, detailing all stages from automatic fault detection to automatic recovery, is elaborated step-by-step in **Algorithm 1**.

5.3.6 ***Future Directions and Knowledge Building.*** The architecture incorporates a fault controller table to build a fault knowledge base. The system can swiftly address recurring faults by storing detailed fault information and corrective actions. For new faults, error virtualization maps the fault to similar known issues for efficient recovery. Checkpoint memory facilitates advanced recovery scenarios, allowing the use of older checkpoints to correct or re-execute faulty operations. This automation ensures the system continuously learns and improves, paving the way for future fault prediction and resolution advancements.

6 Implementation

To validate the efficiency and dependability of the proposed self-healing architecture, we implemented the entire system on an FPGA platform. This implementation integrates CNN-based image processing as the main application and the self-healing architecture, as described in Fig. 5 and Algorithm 1. This implementation illustrates not only the acceleration and energy efficiency introduced by the FPGA hardware accelerators but also the improved system reliability and availability provided by the self-healing mechanisms.

Algorithm 1 Self-Healing Architecture for ADS Perception

***I*:** Input image
***W[l], B[l]*:** Weights and biases for CNN layer l
***M, N, P*:** Matrix indices in each layer l
SyncPoint: Synchronization point for comparison
C[l]: Checkpoints for fault recovery
F.D: Fault Detector for fault detection/diagnosis
F.C.T: Fault Controller Table for fault tracking
R.P: Rescue Point for fault correcting

Receive correctly formatted input data I
Load CNN parameters $W[l], B[l]$ into Applications
while system is running **do**
  for each CNN layer l **do**
    for each matrix element (M, N) **do**
      $Y1_{(\text{Tn})}[M,N] = \sum_P W[l][M,P] \cdot I[l][P,N]$
      $Y2_{(\text{Tn})}[M,N] = \sum_P W[l][M,P] \cdot I[l][P,N]$
      Send $Y1_{(\text{Tn})}$ and $Y2_{(\text{Tn})}$ to SyncPoint
    end for
    Save checkpoints for active applications in (Tn)
    Save Outputs in temp registers $Y1_{(\text{Tn-1})}, Y2_{(\text{Tn-1})}$
    if $Y1_{(\text{Tn})} = Y2_{(\text{Tn})}$ **then**
      Continue normal execution
    else
      comment: Self-Healer functionality
      Trigger **Self-Healer** system
      Delete current checkpoint, including faulty data
      Selecting the best **R.P** for re-execution by Fault controller
      Re-execute $Y1$ and $Y2$ using $C[l]_{\text{last}}$
      if $Y1_{(\text{Tn})} = Y2_{(\text{Tn})}$ **then**
        Classify fault as **transient**
        Resume **normal execution**
        F.C.T $\leftarrow$ Update$(Faulttype, M, N, P, l)$
      else
        Classify fault as **permanent**
        Delete $C[l]_{\text{last}}$
        Selecting best **R.P** for correction
        Activate **Passive** Applications using $C[l]_{\text{last}}$
        Compare $Yp1, Yp2$ with data from temp registers $Y1_{(\text{Tn-1})}, Y2_{(\text{Tn-1})}$
        $(M, N, P, l) \leftarrow$ **F.D**$(C[l], W[l], B[l])$
        FCT $\leftarrow$ Update$(Faulttype, M, N, P, l)$
        Retrieve and Correct Faulty data:

$$W[l][M,N] \leftarrow W_{\text{Passive}}[M,N], \quad B[l][P] \leftarrow B_{\text{Passive}}[P]$$

F.C.T update: Fault$[M, N, P] \rightarrow$ Resolved

      end if
    end if
  end for
  Periodically discard old checkpoints (FIFO strategy)
end while

6.0.1 System Implementation. The CNN model was implemented on an FPGA using VHDL at the Register-transfer level (RTL) to leverage the FPGA's inherent parallelism for real-time processing. The model used is LeNet-5, chosen for its simplicity and efficiency, making it well-suited for integration with the self-healing system. LeNet-5 consists of eight layers, including two convolutional layers, two pooling layers, one flattening layer, and three fully connected layers. These layers are structured to perform feature extraction and classification tasks on image data. To meet ADS's real-time requirements, our CNN implementation leverages the inherent parallelism of FPGA hardware, offering flexibility for full parallelism or pipelined execution to balance resource utilization and performance. Unlike conventional approaches that rely on off-chip memory or hard-coded parameters (which significantly increase latency and resource usage), we optimized the design by storing weights and biases in BRAM, minimizing DSP, LUT, and FF usage while ensuring efficient on-chip data access. The CNN operates in a pipelined manner, achieving parallelism within layers for filter computations and across layers for seamless data propagation. For example, during the second iteration of computation, comparison and checkpointing for the first iteration occur simultaneously, reducing the latency overhead of self-healing. Our *Auto-Healer* supports this resource-efficient design, enabling the deployment of four CNN instances on a single FPGA. By storing parameters in BRAM to reduce DSP, LUT, and FF usage, the design allows two active modules to work in a DMR configuration, with two passive modules available for fault-tolerant operations. This scalable and fault-tolerant implementation is well-suited for high-performance ADS tasks.

Additionally, *Auto-Healer* including synchronization points, fault detectors (F.D), fault controller table (F.C.T), rescue points (R.P), and checkpointing mechanisms, was implemented on the FPGA. These components were connected to the CNN layers to maintain fault tolerance and ensure system reliability. The FPGA processed input images sequentially, assuming no faults were found in the input data. The weights and biases for computations were pre-loaded into the local memory of each CNN instance. The system was implemented with four CNN instances: two active modules and two passive modules under normal operation. During execution, the active modules performed inference tasks, and after each MAC operation, the synchronization point checked the pass_flag to determine its status (1 or 0). A pass_flag of 1 indicated no faults, allowing the system to proceed seamlessly. If the pass_flag was 0, the self-healing mechanism was triggered to locate and correct the fault automatically.

6.0.2 Fault Injection and Scenarios. Runtime fault injection was performed during testbench simulations to evaluate the

system's fault tolerance. These included transient faults such as bit flips (SEU) and computational errors (e.g., malfunctioning multipliers or adders), as well as permanent faults such as stuck-at faults. Faults were injected into CNN layers' weights, biases, and operations, including convolutional and FC layers. The following scenarios were evaluated:

- *Single fault in a single layer:* Transient faults in this scenario could arise from computational errors in operators or single or multiple bit-flips in the local memory of a specific active CNN layer during computation. In this case, permanent faults were related either to operators, which could only be masked by switching to the passive application, or permanent bit flips in the local BRAM of an active CNN, which were corrected by the self-healing mechanism.

- *Multiple faults in a single layer:* Multiple transient faults, such as bit flips in local memory or computational errors in operators within a single CNN layer, were introduced to test the system's ability to resolve complex fault scenarios within a layer. Similarly, multiple permanent bit flips in the local BRAM were injected, requiring the self-healing mechanism to isolate and correct the faults while maintaining system availability.

- *Sequential faults across multiple layers:* Faults were injected sequentially across different CNN layers to simulate scenarios where faults appear progressively during the system's runtime. These faults included both transient faults and permanent faults. The self-healing mechanism detected and resolved each fault autonomously, ensuring uninterrupted system operations.

6.0.3 Fault Detection and Correction. In our system, the F.Ds identify whether a fault is transient or permanent using a counter. When a mismatch is first detected at the synchronization point, the counter is set to 0. The F.D then requests the fault controller to select the best R.P for simple re-execution using the last stable checkpoint, incrementing the counter to 1. If equality is restored during re-execution, the fault is classified as transient (e.g., a bit flip or a transient fault in the logic unit), and the re-execution successfully corrects the fault. If the same mismatch persists, the F.D identifies the fault as permanent (e.g., a stuck-at fault in BRAM or a permanent logic unit failure). In such cases, the F.D records the fault information in the F.C.T, requests the fault controller to isolate faulty applications, and selects the best R.P to activate the passive nodes and re-execute the operation using the last stable checkpoint. For stuck-at faults, while the activated passive applications are running, their output is used to identify the faulty data in BRAM, which is then replaced with the correct data from the new active nodes. The F.C.T is updated to reflect the resolved fault status. If the fault is permanent in the logic unit (not a stuck-at-fault in BRAM), the system also generates an alert for the ADS controller to schedule physical maintenance.

Our *Auto-Healer* system provides distinct advantages over TMR. While TMR involves all three redundant modules to be running simultaneously *at all times*, thereby incurring significant latency and power overhead because of continuous invocation. In contrast, *Auto-Healer* activates only two modules simultaneously, keeping the other two passive and invokes them only when *permanent faults are encountered*, such as stuck-at faults or mask logic unit failures. Furthermore, *Auto-Healer* alerts the ADS system to address physical faults, ensuring long-term reliability. This approach balances resource efficiency and fault management while meeting the stringent real-time requirements of ADS applications.

7 Experimental Results

7.1 Experimental Setup

We conduct experiments to assess the effectiveness of our *Auto-Healer* architecture using a Xilinx Virtex® UltraScale™ FPGA platform that integrates CNN-based perception tasks and fault management for latency, power consumption, resource utilization analysis, and reliability.

- **Convolutional Neural Network:** The LeNet-5 model was adopted to handle image processing due to its simplicity, which allows for efficiency in integration with the self-healing system, besides allowing room for any higher-level complexity of models that may be necessary in the future. LeNet-5, containing convolutional, pooling, and fully connected layers, can effectively show both inference acceleration and fault tolerance of the system. Pre-training of the model on the CIFAR-10 dataset had been done to prepare the parameters for inference. At first, post-training quantization was used to make optimum weights and biases, which reduced the overall FPGA resource usage. The described model was implemented at RTL using VHDL language, and pipeline architecture was used at every stage to minimize the latency and increase the throughput. This scalable hardware design ensures that ADS tasks can maintain real-time on-chip processing even when scaled to more complex datasets, such as COCO and KITTI, for training and inference.

- **Dataset:** The CIFAR-10 dataset has been used for evaluating the self-healing system. CIFAR-10, with 60,000 images of resolution 32x32 pixels distributed across 10 classes [33], provides a more complex benchmark for evaluating the system performance. This dataset was selected to demonstrate the architecture's flexibility in handling a range of visual inputs.

- **Experimentation Platform:** The proposed architecture is implemented on the XCVU440-FLGA2892-2-i of the Virtex® UltraScale™ family. It is a high-performance FPGA platform with many computational resources, such as logic elements, DSP slices, and BRAM. The implemented system consumed less than 2% of the total available resources of the FPGA. Hence, most resources can be spared for other ADS pipeline stages like planning and control.

7.2 Experimental Results and Analysis

This section discusses the experimental results that are necessary to validate the efficiency of the proposed self-healing architecture, with negligible overheads towards system latency and power consumption, for increased reliability. All of the evaluations performed in this study were centered around image processing workloads pertaining to the Perception stage of the ADS pipeline, executed on the FPGA platform. The demonstration of CNN-based image processing integrated with the self-healing mechanisms shows the system's capabilities to reduce latency and power consumption while keeping very high reliability, which is crucial in many safety-critical applications.

7.2.1 ***Reliability, Latency, and Power Efficiency.*** These results are indicative of the dual advantages that the proposed architecture brings about: a reduction in latency and power consumption for CNN inference tasks while it ensures a considerable improvement in system reliability due to self-healing mechanisms. Due to intrinsic parallelism in the FPGA, CNN operations such as convolution and pooling can be efficiently executed with low latency and high throughput. Meanwhile, the self-healing components guarantee fault detection, isolation, and correction to be performed automatically without disrupting system operation.

Latency and Power Constraints in ADS Pipeline. While in most ADSes, the dominant sources for system latency and power consumption stem from perception, especially from image processing, DNN-based computations represent a majority of this workload. For instance, detection tasks (DET) on CPUs exhibit a mean latency of 7150 ms, whereas FPGAs reduce this to just 11.2 ms [36], demonstrating the efficiency gains achievable with FPGA implementations [55]. Also, the perception latency shall remain below 50 ms [3, 36], which is only a fraction of the required deadline for ADS decision-making [3]. We note that using FPGAs significantly reduced both power consumption and processing latency during inference operations using DNN benchmarks compared to other processors like CPU/GPU implementation. This improvement stems from the FPGA's ability to execute tasks in parallel and store model parameters on-chip, reducing data transfer overheads and achieving greater efficiency compared to general-purpose processors. These enabled the system to process more frames per second (FPS), which in turn increased object detection and classification accuracy, and other advanced perception capabilities such as traffic light detection and lane identification that are needed for higher ADS levels [37].

FPGA Acceleration and System Advantages. The FPGA implementation of CNN showcased remarkable latency reductions, enabling sub-50 ms latency for image processing tasks [52]. Moreover, the energy efficiency of FPGAs, with power consumption significantly lower than GPUs, aligns with the stringent power budgets of embedded ADS platforms, typically ranging from 10–30 W [36, 54]. This power efficiency facilitates the integration of additional DNN computations into the perception stage without exceeding energy constraints, supporting the scalability required for higher ADS levels [37].

Initially, the baseline performance of the CNN inference on the FPGA was measured respecting image processing; to set the baseline, the metrics, latency, and power consumption were measured in fault-free conditions. Then, we injected faults at runtime to measure the system's fault tolerance. Synchronization points, F.D, F.C.T, R.P, and checkpointing have been analyzed as self-healing mechanisms respecting their impact on system reliability and overhead. From Table 3, it is evident that latency comparisons represent very favorable evidence for FPGA acceleration compared to using a CPU for ADS perception tasks.

Table 3: Comparison of Latency and Power Consumption for CNN Inference on CPU and FPGA

Model	Scenario	Latency (ms)	Speed-up
LeNet-CPU	No-Fault	43.48	-
LeNet-FPGA	No-Fault	0.95	45

Self-Healing Overhead Analysis. Our experimental results showed that the self-healing system maintained real-time performance with negligible overhead relative to the benefits of improved reliability. Fault injection tests, including transient and permanent faults, demonstrated that the self-healing system was able to correct the errors, maintaining system availability in all scenarios. Table 4, 5 demonstrate the latency and power consumption of the whole system and its overhead compared to a CNN implementation without the self-healing mechanism.

As shown in Table 4, the results from our experiments highlight the minimal latency overhead of the self-healing

Table 4: Latency Comparison for CNN Inference on FPGA Augmented with Self-Healer for Transient and Permanent Fault Types, Including Overhead Compared to a Fault-Free CNN on FPGA Without Fault Tolerance

Scenario	Fault Type	Latency (ms)	Overhead (%)
No Fault	None	0.95080	-
Auto-Healer	Transient	0.95084	0.0042
Auto-Healer	Permanent	0.95092	0.0126

mechanism, particularly in scenarios involving transient and permanent faults. The measured overhead is well within acceptable limits for ADS applications and remains below the threshold value typically required for real-time processing in ADS pipelines. This ensures reliable performance without compromising system efficiency or the stringent timing constraints critical to ADS operations.

Table 5: Comparison of Power Consumption and Absolute Overhead for CNN Inference on FPGA

Scenario	Fault Type	Power (W)	Overhead (W)
No-Fault	None	5.00	-
Auto-Healer	Transient	5.50	0.50
Auto-Healer	Permanent	5.80	0.80

As shown in Table 5, our experimental results demonstrate the minimal power overhead introduced by the self-healing mechanism, making it highly suitable for energy-constrained ADS applications. This efficiency enables extended operational time and ensures the integration of fault-tolerant mechanisms without exceeding the stringent power budgets of such systems.

Impact on Higher ADS Levels. The proposed architecture's ability to maintain low latency and high reliability makes it particularly suitable for higher ADS levels. As autonomy increases, so does the demand for advanced DNN computations, such as 360-degree object tracking and SLAM [37]. By accelerating inference and incorporating self-healing mechanisms, the architecture ensures both performance and reliability, overcoming the latency and power challenges associated with advanced ADS tasks.

Table 6 and Table 7 show the FPGA resource utilization for implementing the CNN without and with self-healing.

The self-healing system introduces minimal overhead, with LUT utilization increasing from 0.35% to 1.91% and BRAM usage from 0.65% to 2.54%, ensuring enhanced reliability with negligible resource impact.

7.2.2 ***Reliability Metrics Analysis.*** We study reliability metrics, such as Mean Time to Repair (MTTR) and Mean Time to Detect (MTTD), to evaluate the fault tolerance and dependability of the proposed self-healing architecture for ADS. These metrics quantify the system's capability to detect faults promptly and restore functionality, ensuring high availability and fault resilience in safety-critical scenarios [25, 40, 65]. MTTD measures the average time required to detect and classify a fault, while MTTR reflects the time needed to repair the fault after detection, encompassing isolation, correction, and recovery processes.

Table 6: FPGA Resource Utilization for CNN

Resource Type	Available	Utilized	Utilization (%)
LUT	2532960	8894	0.35
LUTRAM	459360	1020	0.22
FF	5065920	11372	0.22
DSP	2880	0	0
BRAM	2520	17	0.65
I/O	1456	120	8.24

Table 7: FPGA Resource Utilization for the Whole System

Resource Type	Available	Utilized	Utilization (%)
LUT	2532960	48384	1.91
LUTRAM	459360	4449	0.97
FF	5065920	55548	1.10
DSP	2880	0	0
BRAM	2520	64	2.54
I/O	1456	122	8.38

Results and Analysis. The experimental results validated the effectiveness of the self-healing architecture. Table 8 summarizes the detection and repair times for transient and permanent faults under various scenarios.

Table 8: Reliability Metrics Across Fault Scenarios

Metric	Transient Fault	Permanent Fault
MTTD (ns)	20	60
MTTR (ns)	20	60
Total (ns)	40	120

The metrics presented in Table 8 illustrate that transient faults were detected and repaired within a total of 40 ns, while permanent faults required 120 ns. The architecture demonstrated robust performance across all scenarios, maintaining low detection and repair times even under challenging conditions. To evaluate the self-healing mechanism's

impact on system performance, the latency overhead was calculated by comparing the normal CNN execution time (0.95080 ms) with the execution time under fault conditions with healing. For transient faults, the healing process introduced a latency overhead of approximately 0.0042%, while for permanent faults, the overhead was 0.0126%. These negligible overheads demonstrate the efficiency of the self-healing architecture in maintaining real-time operation while correcting faults.

Our experimental results underscore the architecture's capability to handle both transient and permanent faults effectively, paving way toward compliance for stringent safety and reliability requirements of ADS standards like ASIL-D.

8 Related Work

Baleani et al. [6] presented fault tolerance architecture performance in single-chip multi-core processors targeting safety-critical automotive applications. Stoffel and Sax [53] presented "Voter-as-a-Service", a framework using distributed redundant copies of automotive applications over multi-ECUs interconnected using LAN, and therefore resilient to system-level faults. Li and Song [9] implemented an ADS design featuring three subsystems: braking, driving, and steering, along with fault tolerance algorithms. Likewise, fault tolerance methods that implement an urban autonomous-driving system are also realized through the Stadtpilot by Volkswagen and showed practical realization potential for such systems [30, 41]. These represent the growing emphasis being placed on robust ADS designs considering sensor failures, computation errors, and control inaccuracies. In the domain of Distributed Fault-Tolerant Systems, Ishigooka et al. [26] present different architecture-level designs to attain fail-operational capabilities of autonomous driving by replicating critical operations to handle the faults with higher efficiency. Similarly, Portet et al. [43] explored software-only TMR approaches for commercial-off-the-shelf GPUs, demonstrating that execution staggering has the potential to meet standards such as ASIL-D. Meanwhile, in [44], Poudel et al. compared the dependability of the GPGPU- and FPGA-based MPSoC ECU architectures using both redundant multi-threading and self-reconfigurable DMR for X-by-wire systems in ADS.

In the context of fault tolerance of CNNS, Zhao et al. [67] developed a new approach to fault-tolerant CNNs, dubbed FT-CNN, which deploys ABFT schemes in order to perform robust CNN inference. FT-CNN also supports bias operations, grouped convolution, and backpropagation, therefore extending to a wide range of configurations. However, it relies so much on the IntelCaffe framework and assumes single-fault model limitations to complex, realistic conditions in ADS applications, for instance. These impose requirements for further innovations within such methods to handle and further optimize reliability in multi-directives, hardware-specific topological architectures such as FPGA-Based, specifically for ADS Pipelines.

Recently, a novel FPGA-based CNN accelerator proposed in Syed et al. [56] includes fault tolerance and reconfigurability as supportive technologies for multi-modal multi-task applications. Their design, by the use of shared-layer methodologies, suits every one of the operational modes required: reliability-optimizing [31], performance-optimizing, and energy-efficient-optimizing. They demonstrate how effective Triple Modular Redundancy really is for error resilience in combination with pruning and quantization to bring hardware-resource utilization down by at least an order of magnitude. While their approach yields impressive results in terms of accuracy and performance on FPGAs, its implications on other platforms and in real-time systems, such as ADS, have been less explored. More opportunities for further advancements would be opened up in safety-critical and latency-sensitive areas.

9 Conclusion

This paper proposes a self-healing hardware architecture for enhancing reliability during the perception stage of an ADS system. Leveraging DMR and FPGA-based reconfigurability, the system ensures seamless automatic fault detection and correction. Additional energy consumption is minimized by activating the self-healing module only upon detected faults, facilitated by lightweight synchronization mechanisms. Our experimental results demonstrate a significant increase in fault tolerance and reliability, achieving a latency of 0.95092 ms, power consumption of 5.8 W, area utilization of 1.91% LUTs and 2.54% BRAMs, and exceptionally low MTTR of 40 ns for transient faults and 120 ns for permanent faults. These results validate the effectiveness of the architecture in safety-critical ADAS applications while meeting stringent real-time operational constraints.

Acknowledgments

This work is supported by the Office of Naval Research under Grant N00014-24-1-2046.

References

[1] Udit Kumar Agarwal, Abraham Chan, Ali Asgari, and Karthik Pattabiraman. 2023. Towards Reliability Assessment of Systolic Arrays against Stuck-at Faults. In *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*. IEEE, 230–236.

[2] Mohammad Hasan Ahmadilivani, Seyedhamidreza Mousavi, Jaan Raik, Masoud Daneshtalab, and Maksim Jenihhin. 2024. Cost-Effective Fault Tolerance for CNNs Using Parameter Vulnerability Based Hardening and Pruning. In *2024 IEEE 30th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, 1–7. https://doi.org/10.1109/IOLTS60994.2024.10616072

[3] Miguel Alcon, Hamid Tabani, Leonidas Kosmidis, Enrico Mezzetti, Jaume Abella, and Francisco J Cazorla. 2020. Timing of autonomous driving software: Problem analysis and prospects for future solutions. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 267–280.

[4] Audi, Intel, Volkswagen, and Others. 2019. Safety First for Automated Driving. https://www.aptiv.com/en/newsroom/article/automotive-and-mobility-industry-leaders-publish-first-of-its-kind-framework-for-safe-automated-driving-systems. Accessed: 2024-12-18.

[5] Baidu Apollo. 2018. Baidu Apollo: An Open Autonomous Driving Platform. http://apollo.auto/. Accessed: 2024-12-18.

[6] Massimo Baleani, Alberto Ferrari, Leonardo Mangeruca, Alberto Sangiovanni-Vincentelli, Maurizio Peri, and Saverio Pezzini. 2003. Fault-tolerant platforms for automotive safety-critical applications. In *Proceedings of the 2003 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*. ACM, 170–177.

[7] David J Bryant. 2006. Rethinking OODA: Toward a modern cognitive framework of command decision making. *Military Psychology* 18, 3 (2006), 183–206.

[8] Cadence Design Systems. 2024. Functional Safety Methodologies for Automotive Applications. https://www.multimediadocs.com/assets/cadence_emea/documents/functional_safety_methodologies_for_automotive_applications.pdf. Accessed: 2024-12-18.

[9] Li DanYong and Song YongDuan. 2012. Adaptive fault-tolerant tracking control of 4WS4WD road vehicles: A fully model-independent solution. In *Proceedings of the 31st Chinese Control Conference*. IEEE, 485–492.

[10] Preet Derasari, Kailash Gogineni, and Guru Venkataramani. 2023. Mayalok: A cyber-deception hardware using runtime instruction infusion. In *2023 IEEE 34th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 33–40.

[11] Preet Derasari, Kailash Gogineni, and Guru Venkataramani. 2023. Mayavi: A cyber-deception hardware for memory load-stores. In *Proceedings of the Great Lakes Symposium on VLSI 2023*. Association for Computing Machinery, 563–568.

[12] Preet Derasari and Guru Venkataramani. 2024. Maya: Hardware Enhanced Customizable Defenses at the User-Kernel Interface. In *2024 International Symposium on Secure and Private Execution Environment Design (SEED)*. IEEE, 50–61.

[13] Javier Fernández, Jon Perez, Irune Agirre, Imanol Allende, Jaume Abella, and Francisco J Cazorla. 2021. Towards Functional Safety Compliance of Matrix–Matrix Multiplication for Machine Learning-Based Autonomous Systems. *Journal of Systems Architecture* 121 (2021), 102298.

[14] Zhen Gao, Han Zhang, Yi Yao, Jiajun Xiao, Shulin Zeng, Guangjun Ge, Yu Wang, Anees Ullah, and Pedro Reviriego. 2022. Soft error tolerant convolutional neural networks on FPGAs with ensemble learning. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 30, 3 (2022), 291–302.

[15] Joshua Garcia, Yang Feng, Junjie Shen, Sumaya Almanee, Yuan Xia, Chen, and Qi Alfred. 2020. A comprehensive study of autonomous vehicle bugs. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. ACM, 385–396.

[16] Florian Geissler, Syed Qutub, Sayanta Roychowdhury, Ali Asgari, Yang Peng, Akash Dhamasia, Ralf Graefe, Karthik Pattabiraman, and Michael Paulitsch. 2021. Towards a safety case for hardware fault tolerance in convolutional neural networks using activation range supervision. *arXiv preprint arXiv:2108.07019* (2021). https://arxiv.org/abs/2108.07019

[17] Kailash Gogineni, Yongsheng Mei, Karthikeya Gogineni, Peng Wei, Tian Lan, and Guru Venkataramani. 2024. Characterizing and Optimizing the End-to-End Performance of Multi-Agent Reinforcement Learning Systems. In *2024 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 224–235.

[18] Mukul Anil Gosavi, Benjamin B Rhoades, and James M Conrad. 2018. Application of functional safety in autonomous vehicles using ISO 26262 standard: A survey. In *SoutheastCon 2018*. IEEE, 1–6.

[19] Pamela M Greenwood, John K Lenneman, and Carryl L Baldwin. 2022. Advanced Driver Assistance Systems (ADAS): Demographics, Preferred Sources of Information, and Accuracy of ADAS Knowledge. *Transportation Research Part F: Traffic Psychology and Behaviour* 86 (2022), 131–150.

[20] Dominique Gruyer, Valentin Magnier, Karima Hamdi, Laurène Claussmann, Olivier Orfila, and Andry Rakotonirainy. 2017. Perception, information processing and modeling: Critical stages for autonomous driving applications. *Annual Reviews in Control* 44 (2017), 323–341.

[21] Muhammad Abdullah Hanif and Muhammad Shafique. 2020. Dependable deep learning: Towards cost-efficient resilience of deep neural network accelerators against soft errors and permanent faults. In *2020 IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, 1–4.

[22] Peter E Hart, Nils J Nilsson, and Bertram Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4, 2 (1968), 100–107.

[23] Yi He, Mike Hutton, Steven Chan, Robert De Gruijl, Rama Govindaraju, Nishant Patil, and Yanjing Li. 2023. Understanding and mitigating hardware failures in deep learning training systems. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 1–16.

[24] Christina Houben and Sebastian Houben. 2015. Endowing advanced driver assistance systems with fault tolerance. *Annual Reviews in Control* 39 (2015), 58–67.

[25] HamidReza Imani, Jeff Anderson, Samuel Farid, Abdolah Amirany, and Tarek El-Ghazawi. 2024. RLFL: A Reinforcement Learning Aggregation Approach for Hybrid Federated Learning Systems Using Full and Ternary Precision. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* (2024).

[26] Tasuku Ishigooka, Shinya Honda, and Hiroaki Takada. 2018. Cost-effective redundancy approach for fail-operational autonomous driving system. In *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*. IEEE, 107–115.

[27] Leonardo Iurada, Niccolò Cavagnero, Fernando Fernandes Dos Santos, Giuseppe Averta, Paolo Rech, and Tatiana Tommasi. 2024. Transient Fault Tolerant Semantic Segmentation for Autonomous Driving. *arXiv preprint* arXiv:2408.16952 (2024). Preprint available at https://arxiv.org/abs/2408.16952.

[28] Adam Jacobs, Grzegorz Cieslewski, Alan D George, Ann Gordon-Ross, and Herman Lam. 2012. Reconfigurable fault tolerance: A comprehensive framework for reliable and adaptive FPGA-based space computing. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 5, 4 (2012), 1–30.

[29] Belal Jahannia, Abdolah Amirany, Elham Heidari, and Hamed Dalir. 2025. DaLAMED: A Clock-Frequency and Data-Lifetime-Aware Methodology for Energy-Efficient Memory Design in Edge Devices. *IEEE Access* (2025).

[30] Ehsan Mousavi Khaneghah, Amirhosein Reyhani ShowkatAbad, Nosratollah Shadnoush, Nigar Ismayilova, Reyhaneh Noorabad Ghahroodi, Elviz Ismayilov, Mohammad Saeed Nabati Saravani, Fatemeh Taheri Sarraf, and Ali Soveizi. 2018. ExaMig matrix: Process migration based on matrix definition of selecting destination in distributed exascale environments. *Azerbaijan Journal of High Performance Computing* 1, 1 (2018), 20–41.

[31] Farid Kochakkashani, Vahid Kayvanfar, and Roberto Baldacci. 2024. Innovative Applications of Unsupervised Learning in Uncertainty-Aware Pharmaceutical Supply Chain Planning. *IEEE Access* 12 (2024), 107984–107999. https://doi.org/10.1109/ACCESS.2024.3435439

[32] Leonidas Koutras and Zoe Doulgeri. 2020. Dynamic Movement Primitives for Moving Goals with Temporal Scaling Adaptation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 144–150.

[33] A. Krizhevsky. 2009. The CIFAR-10 dataset. Online. Available: http://www.cs.toronto.edu/~kriz/cifar.html.

[34] Maya Kumar and Berend van der Kolk. 2020. Audi A8: The World's First Level 3 Autonomous Vehicle. https://hbsp.harvard.edu/product/W20134-PDF-ENG. Accessed: 2024-12-18.

[35] LeddarTech. 2024. An Explanation of Perception Performance Paradigm. https://leddartech.com/app/uploads/dlm_uploads/2024/01/White-Paper_An-Explanation-of-Perception-Performance-Paradigm_V1.0_EN.pdf. Accessed: 2024-12-18.

[36] Shih-Chieh Lin, Yunqi Zhang, Chang-Hong Hsu, Matt Skach, Md E Haque, Lingjia Tang, and Jason Mars. 2018. The Architectural Implications of Autonomous Driving: Constraints and Acceleration. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 751–766.

[37] Shaoshan Liu, Jie Tang, Zhe Zhang, and Jean-Luc Gaudiot. 2017. Computer architectures for autonomous driving. *Computer* 50, 8 (2017), 18–25.

[38] Mobileye. 2024. Understanding L2+ in Five Questions. https://www.mobileye.com/blog/understanding-l2-in-five-questions/. Accessed: 2024-12-18.

[39] Iraj Moghaddasi, Saeid Gorgin, and Jeong-A Lee. 2023. Dependable DNN Accelerator for Safety-Critical Systems: A Review on the Aging Perspective. *IEEE Access* 11 (2023), 89803–89834. https://doi.org/10.1109/ACCESS.2023.3300376

[40] National Institute of Standards and Technology (NIST). 2006. Performance Metrics for Intelligent Systems (PerMIS) Workshop. http://dx.doi.org/10.6028/NIST.SP.1062. Workshop held in Gaithersburg, Maryland, USA, August 21-23, 2006. Co-located with the IEEE International Workshop on Safety, Security, and Rescue Robotics (SSRR).

[41] Tobias Nothdurft, Peter Hecker, Sebastian Ohl, Falko Saust, Markus Maurer, Andreas Reschka, and Jürgen Rüdiger Böhmer. 2011. Stadtpilot: First fully autonomous test drives in urban traffic. In *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 919–924.

[42] Optima Design Automation. 2019. ISO 26262 Primer: White Paper. https://www.optima-da.com/wp-content/uploads/2019/10/Optima-ISO-26262-Primer-White-Paper-191028.pdf. Accessed: 2024-12-18.

[43] Sergi Alcaide Portet, Leonidas Kosmidis, Carles Hernandez, and Jaume Abella. 2020. Software-only triple diverse redundancy on GPUs for autonomous driving platforms. In *2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*. IEEE, 82–88.

[44] Bikash Poudel, Naresh Kumar Giri, and Arslan Munir. 2017. Design and comparative evaluation of GPGPU-and FPGA-based MPSoC ECU architectures for secure, dependable, and real-time automotive CPS. In *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 29–36.

[45] Taufiq Rahman, Andrew Liu, Daniel Cheema, Victor Chirila, and Dominique Charlebois. 2023. ADAS Reliability Against Weather Conditions: Quantification of Performance Robustness. In *27th International Technical Conference on the Enhanced Safety of Vehicles (ESV) National Highway Traffic Safety Administration*. National Highway Traffic Safety Administration (NHTSA), 306–310.

[46] George A Reis, Jonathan Chang, Neil Vachharajani, Ram Rangan, and David I August. 2005. SWIFT: Software implemented fault tolerance. In *International Symposium on Code Generation and Optimization*. IEEE, 243–254.

[47] Mohammad Hadi Rezayati, Abdolah Amirany, Mohammad Hossein Moaiyeri, and Kian Jafari. 2025. A new method for securing binary deep neural networks against model replication attacks using magnetic tunnel junctions. *International Journal of Information Security* 24, 1 (2025), 1–16.

[48] SAE International. 2014. AUTOMATED DRIVING, Levels of driving automation are defined in new SAE International standard J3016. http://www.sae.org/misc/pdfs/automated_driving.pdf.

[49] Bülent Sari. 2020. *Fail-operational Safety Architecture for ADAS/AD Systems*. Springer.

[50] John M Scanlon, Kristofer D Kusano, Laura A Fraade-Blanar, Timothy L McMurry, Yin-Hsiu Chen, and Trent Victor. 2024. Benchmarks for retrospective automated driving system crash rate analysis using police-reported crash data. *Traffic Injury Prevention* 25, 1 (2024), 1–15. https://doi.org/10.1080/15389588.2024.2380522

[51] Max Schwenzer, Muzaffer Ay, Thomas Bergs, and Dirk Abel. 2021. Review on model predictive control: An engineering perspective. *The International Journal of Advanced Manufacturing Technology* 117, 5 (2021), 1327–1349.

[52] Omais Shafi, Chinmay Rai, Rijurekha Sen, and Gayathri Ananthanarayanan. 2021. Demystifying TensorRT: Characterizing Neural Network Inference Engine on NVIDIA Edge Devices. In *2021 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 226–237. https://doi.org/10.1109/IISWC53511.2021.00030

[53] Martin Stoffel and Eric Sax. 2023. Distributed Voters for Automotive Applications. In *2023 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 1–8. https://doi.org/10.1109/IV55152.2023.10186579

[54] Ali Suvizi, Azim Farghadan, and Morteza Saheb Zamani. 2023. A parallel computing architecture based on cellular automata for hydraulic analysis of water distribution networks. *J. Parallel and Distrib. Comput.* 178 (2023), 11–28. https://doi.org/10.1016/j.jpdc.2023.03.009

[55] Ali Suvizi, Suresh Subramaniam, Tian Lan, and Guru Venkataramani. 2024. Exploring In-Memory Accelerators and FPGAs for Latency-Sensitive DNN Inference on Edge Servers. In *2024 IEEE Cloud Summit*. IEEE, Arlington, VA, USA, 1–6. https://doi.org/10.1109/CloudSummit61220.2024.00007

[56] Rizwan Tariq Syed, Yanhua Zhao, Junchao Chen, Marko Andjelkovic, Markus Ulbricht, and Milos Krstic. 2024. FPGA Implementation of a Fault-Tolerant Fused and Branched CNN Accelerator With Reconfigurable Capabilities. *IEEE Access* 12 (2024), 57847 – 57862. https://doi.org/10.1109/ACCESS.2024.3392240

[57] Hamid Tabani, Roger Pujol, Jaume Abella, and Francisco J Cazorla. 2020. A Cross-Layer Review of Deep Learning Frameworks to Ease Their Optimization and Reuse. In *2020 IEEE 23rd International Symposium on Real-Time Distributed Computing (ISORC)*. IEEE, 144–145. https://doi.org/10.1109/ISORC49007.2020.00030

[58] Emil Talpes, Debjit Das Sarma, Ganesh Venkataramanan, Peter Bannon, Bill McGee, Benjamin Floering, Ankit Jalote, Christopher Hsiong, Sahil Arora, Atchyuth Gorti, et al. 2020. Compute solution for Tesla's full self-driving computer. *IEEE Micro* 40, 2 (2020), 25–35.

[59] The Autoware Foundation. 2016. Autoware: An Open Autonomous Driving Platform. https://github.com/CPFL/Autoware/. Accessed: 2024-12-18.

[60] The New York Times. 2019. Self-Driving Uber Car Kills Pedestrian in Arizona, Where Robots Roam. https://www.nytimes.com/2018/03/19/technology/uber-driverless-fatality.html. Accessed: 2024-12-18.

[61] Ivan Todorović, Ivana Isakov, and Marko Gecic. 2023. Development of Electric Vehicles Applications Using AURIX™ Microcontroller and Typhoon HIL. In *Real-Time Simulation and Hardware-in-the-Loop Testing Using Typhoon HIL*. Springer, 157–186.

[62] Abdullah Turan. 2024. PID controller design with a new method based on proportional gain for cruise control system. *Journal of Radiation Research and Applied Sciences* 17, 1 (2024), 100810.

[63] U.S. Department of Transportation – Federal Highway Administration. 2021. Highway Statistics: Status of the Nation's Highways, Bridges, and Transit: Conditions & Performance Report. https://www.fhwa.dot.gov/policy/24cpr/ Accessed: [Insert Date Here].

[64] U.S. Department of Transportation, National Highway Traffic Safety Administration. 2017. Federal Automated Vehicles Policy: Accelerating the Next Revolution in Roadway Safety. https://www.transportation.gov/AV.

[65] Seongwoo Woo. 2020. Modern Definitions in Reliability Engineering. In *Reliability Design of Mechanical Systems*. Springer, Singapore, 53–99. https://doi.org/10.1007/978-981-13-7236-0_3

[66] Zheng Xu and Jacob Abraham. 2019. Safety design of a convolutional neural network accelerator with error localization and correction. In *Proceedings of the 2019 IEEE International Test Conference (ITC)*. IEEE, Washington, DC, USA, 1–10.

[67] Kai Zhao, Sheng Di, Sihuan Li, Xin Liang, Yujia Zhai, Jieyang Chen, Kaiming Ouyang, Franck Cappello, and Zizhong Chen. 2020. FT-CNN: Algorithm-based fault tolerance for convolutional neural networks. *IEEE Transactions on Parallel and Distributed Systems* 32, 7 (2020), 1677–1689.