Exploiting Temporal Data Diversity for Detecting Safety-critical Faults in AV Compute Systems

Saurabh Jha*†‡, Shengkun Cui*†, Timothy Tsai†, Siva Kumar Sastry Hari†, Michael B. Sullivan†,
Zbigniew T. Kalbarczyk*, Stephen W. Keckler† and Ravishankar K. Iyer*
*UIUC, †NVIDIA, ‡IBM

***Abstract*—Silent data corruption caused by random hardware faults in autonomous vehicle (AV) computational elements is a significant threat to vehicle safety. Previous research has explored design diversity, data diversity, and duplication techniques to detect such faults in other safety-critical domains. However, these are challenging to use for AVs in practice due to significant resource overhead and design complexity. We propose, DiverseAV, a low-cost data-diversity-based redundancy technique for detecting safety-critical random hardware faults in computational elements. DiverseAV introduces data-diversity between the redundant agents by exploiting the temporal semantic consistency available in the AV sensor data. DiverseAV is a black-box technique that offers a plug-and-play solution as it requires no knowledge of the internals of the AI agent responsible for executing driving decisions, requiring little to no modification to the agent itself for achieving high coverage of transient and permanent hardware faults. It is commercially viable because it avoids software modifications to agents that are costly in terms of development and testing time. Specifically, DiverseAV distributes the sensor data between the two software agents in a round-robin manner. As a result, the sensor data for two consecutive time steps are semantically similar in terms of their worldview but significantly different at the bit level, thus ensuring the state and data diversity between the two agents necessary for detecting faults. We demonstrate DiverseAV using an open-source self-driving AI agent which is controlling a car in an open-source world simulator.**

I. Introduction

Random hardware faults, transient or permanent [1], caused by power-supply spikes, electrostatic discharge, and external radiation strikes in the computational elements, such as CPUs, GPUs, and ASICs, used in autonomous vehicles (AVs) pose a significant threat to the safety of the autonomous vehicles [2], [3]. Such faults may lead to a detectable uncorrectable error (DUE) that degrades system availability or an undetected error, i.e., a silent data corruption (SDC), that may cause vehicle misbehavior. Practical implementations of autonomous driving systems include a fail-back system that maintains the safety of the system in the case of a DUE. In contrast, faulty behavior due to an SDC may lead to significant safety hazards, resulting in loss of human life and serious damage to vehicles [4]–[6]. Future trends of increasing code complexity and shrinking feature sizes will only contribute to increasing failure rates, thereby exacerbating the problem.

Current strategies for error mitigation include fault avoidance and error detection mechanisms. Often DRAM memories and large SRAM arrays are protected by ECC (error correcting codes), data communication is protected by parity and checksums [7], [8], and critical software, including operating systems, are executed on duplicated cores. Other techniques include assertions [9]–[12], duplication [13], [14], [14]–[20], and data and design diversity techniques [21]–[23] that can be employed at the hardware or software level. Although these strategies are largely effective, they are challenging to use in practice because of chip area and power overheads, performance degradation, and design complexity. Furthermore, due to the significant probability that an error in the AV software is masked [6], many instances of error detection degrade the AV system availability without improving system safety.

Our Approach. We address the above-mentioned challenges by proposing DiverseAV, a lightweight, software-based redundancy technique that exploits the temporal data diversity present in the sensor data for detecting hardware faults. It is an affordable alternative to a fully duplicated system for detecting transient and permanent hardware faults. DiverseAV trades off a marginal decrease (as compared with fully duplicated system) in error detection coverage to achieve significantly lower design complexity and resource overheads along with corresponding power savings. DiverseAV incorporates the following key ideas.

Independent and data-diverse agents. DiverseAV instantiates two independent software processes that use the same autonomous vehicle software code (referred to as an AI-agent or agent). However, unlike a fully duplicated system in which agents use the exact same sensor data, DiverseAV introduces data diversity between the two agents in which the two agents use diverse data to compute control decisions. DiverseAV introduces data diversity by exploiting the temporal semantic consistency in the autonomous vehicle workload: the sensor data (e.g., two subsequent video frames from camera sensors that provide an object's location and environment state) do not change significantly from one time step to another (e.g., from one time frame to another) even though the data at the bit level (i.e., bit/pixel level representation of a camera image/frame) is vastly different. DiverseAV does so by distributing the sensor data in a round-robin manner between the two redundant agents, thereby, introducing data diversity between two consecutive time steps at the instruction level while maintaining the semantic consistency between the two agents at the world level. The data-diverse agents produce similar but not necessarily the same output and the divergence between the two outputs in a fault-free execution is bounded due to the semantic similarity of the inputs in adjacent time steps.

Because much of the data processing in each agent depends on the input data rate, each agent receives half the data and requires roughly half the compute resources. However, the question is: Does this decrease in the input data rate per agent lead to safety hazards? Recent work [24] on AVs (e.g., NVIDIA DriveAV [25]) shows that the AVs can often tolerate a significant drop in the input data rate without causing safety hazards. For example, authors show that the camera sensing rate can be decreased from 30 to 10 Hz safely for many scenarios.

Fault propagation and error detection. Since the divergence between the outputs of the two data-diverse agents is bounded in the fault-free case, DiverseAV is able to detect the error by

comparing the output actuator commands (brake, throttle, and steering angle commands) of the agents. Most errors in AV software state do not propagate in a way that significantly affects actuator commands [6]. For those errors that do significantly affect actuator commands, the outputs of the two agents may diverge depending on the fault type, propagation, and masking in each of the individual processes. (i) A transient fault affects *only* one process enabling DiverseAV to detect the fault because of the independence between the agents. The fault-free agent produces fault-free outputs whereas the other agent (impacted by the fault) produces the corrupted outputs. (ii) A permanent fault that affects both processes may be detectable because in the presence of a fault the two agents produce significantly different corrupted outputs as the corruption depends on the internal (private) state of and inputs to each agent, which are diverse by design [23]. There is a non-zero chance that a permanent fault corrupts the state of both agents in a way that results in sufficiently small output divergence; thereby evading detection. Such permanent faults can be detected using a fully duplicated system but not DiverseAV. Thus, DiverseAV trades off error detection coverage for lower overheads, design complexity and availability. However, our results show that the probability of a missed fault resulting in a safety hazard is small.

Overall, DiverseAV is a black-box technique that offers a plug-and-play solution as it requires no knowledge of the internals of the Autonomous driving system (ADS), requiring little to no modification to the agent itself for achieving high coverage of transient and permanent hardware faults. It is commercially viable because it avoids software modifications to agents that are costly in terms of development and testing time. It is advantageous as it provides the state diversity between the two agents needed to detect random hardware faults at a significantly lower cost, thus, eliminating the need for a fully duplicated system. In this paper, we focus on self-driving cars as our target autonomous system as they operate in highly complex and dynamic environment consisting of pedestrians and other human-controlled actors.

Contributions. Our contributions include the following:

(i) We propose a novel high fault detection coverage and low overhead design called DiverseAV for detecting random hardware faults, transient and permanent, in the compute elements of the AV.

(ii) We have implemented the proposed design using an open-source agent [26] and an open-source simulation platform [27].

(iii) We provide an empirical characterization of temporal data diversity in onboard sensors.

(iv) Using the open-source fault injection tools `NVBitFI` [28] and `PinFI` [29][1], we have performed an experimental assessment of the functional safety of DiverseAV in fault-free operation and in the presence of faults.

(v) Finally, we compare the fault detection capabilities of DiverseAV-enabled ADS with an ADS that uses (i) full duplication of the agents to detect mismatch in outputs of the agents, and (ii) temporal outlier detection on the outputs of a single agent to detect anomalies. In each case, the underlying agent is the same.

Results. Key results include the following:

[1]In [30], authors have shown that fault injection-based fault simulation techniques estimate SDC FIT rates that are comparable with beam test results.

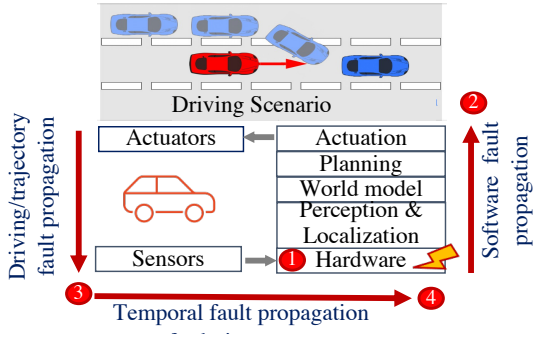


Figure 1: Depiction of fault propagation.

(i) *High safety.* DiverseAV did not lead to safety hazards in any of the evaluated driving scenarios. The trajectory of the vehicle when using DiverseAV closely follows the trajectory of a single-agent-driven system. We found the maximum longitudinal difference between the vehicle positions for two trajectories (i.e., those of a single agent versus the two-agent system configuration) to be <50cm, which is significantly less than the safety distance (5m) that must be maintained by the agent while driving.

(ii) *Highly accurate.* DiverseAV detected safety-critical errors caused by the transient and permanent faults injected into the compute elements (CPUs and GPUs). For GPUs, DiverseAV achieved a precision of 0.87 and a recall of 0.87[2]. For CPUs, all our injected faults were either masked or resulted in a DUE, and we expect the contribution of CPU faults to the system-level SDC FIT rate to be low. Depending on the platform's ability to detect DUEs, DiverseAV achieved a precision of 1.0 and recall of 1.0. Across all our driving scenarios, DiverseAV did not raise an alarm (i.e., detected an error) for fault-free experimental runs. DiverseAV outperforms a single-agent system (which uses temporal outlier detection techniques) in terms of accuracy. The single-agent system yields a precision and recall of 0.17 and 0.52, respectively. We assume availability of a fail-back system that can be invoked on error to bring the vehicle to a safe state.

(iii) *Low performance overhead.* Compared to a fully duplicated system, DiverseAV reduced the resource demands so that the same processor provisioned for a single-agent system is sufficient to handle DiverseAV's two agents, although twice as much memory is needed to accommodate the two independent agents. In contrast, a fully duplicated system requires double the number of processors and memory. DiverseAV trades off a marginal decrease in error detection coverage (compared to fully duplicated system) for reduction in compute resource overhead and design complexity while increasing availability.

II. Faults and their Implication on Safety

A. Fault Models

Faults are the primary concern of ISO 26262 [2], which is an important certification standard for AV compute systems. In this paper, we *only* consider random hardware faults, transient or permanent, that occur in the computational elements of

[2]Higher the precision lower is the number of false alarms (i.e., raising an alarm for an existence of a safety critical fault even when there is no safety violation), whereas higher the recall (also known as sensitivity) lower is the number of safety critical faults that are missed by DiverseAV.

the processor, including pipeline stages, flip-flops, arithmetic and logic units (ALUs), and the register file. We do not consider faults in the main memory or cache, as we assume that these are protected with ECC (e.g., NVIDIA GPUs [31]). However, when the random hardware faults propagate, they eventually corrupt the destination register of the executing opcode [28], [29], [31]–[33]. We emulate these faults via instruction-level bit-flip models, in the computational fabrics used by the ADS (such as CPUs and GPUs), using state-of-the-art fault injection tools [29], [34]. We do not consider sensor faults (e.g., corrupted video frame due to a camera sensor failure) or machine-learning inference faults (e.g., due to out-of-distribution data).

In the transient fault model, we emulate the effect of a fault by corrupting the destination register of *only* one dynamic instruction[3]. In contrast, in the permanent fault model, we emulate the effect of the fault by corrupting the destination register of a selected opcode for *all* dynamic instances of that opcode. The destination register is corrupted by XOR-ing the original contents of the destination register with a selected mask. In this work, we *only* aim to detect faults (transient or permanent) that lead to safety hazards and *do not* aim to identify the fault type (i.e., differentiate transient from permanent fault).

B. Impact of Faults on Safety

Hardware faults can alter the actuation decision outputs, thereby impacting the safety of the vehicle. A typical hardware fault propagation path is shown in Fig. 1. Hardware faults may corrupt the output of the hardware instruction (e.g., output of the add instruction), which in turn can corrupt the output of the software module (e.g., perception outputs). The corrupted values are then consumed by other software modules; which may ultimately taint the actuation outputs. The fault propagation in the software may also corrupt the internal state of the software (until the next reset/restart), which may result in subsequent corruption of actuation outputs in the future time steps. The corrupted actuation outputs for one or more time steps may accumulate and change the vehicle's kinematics sufficiently to cause an accident. Not all faults are hazardous to the system. Faults may result in benign masking, a silent data corruption (SDC), a hang, or a crash. Hangs and crashes are detected by the system via exceptions and heartbeats, whereas SDCs may potentially propagate to cause safety violations. Lockstep-based full duplication of software and hardware ensures robust detection of SDCs; however, they result in high resource provisioning, power overheads, and design complexity [15]. The impact of random hardware faults on safety is discussed through extensive experiments in §V-C.

III. Methodology and Approach

A. Design Requirements

DiverseAV addresses the following design requirements.

Detection of transient and permanent faults. DiverseAV must *only* detect transient and permanent faults that are safety-critical with high probability sufficiently far in advance. Detecting faults masked by hardware or software will significantly reduce the overall system availability.

Driving scenario-independent. The error detector in DiverseAV must be able to detect errors for all possible driving scenarios and should not be limited to only those driving scenarios that were used to train the error detector model.

Plug and play design. DiverseAV must implement a plug-and-play design, thereby reducing the engineering and deployment effort. This means that the AV agent code can be treated as a black box, where only the API for inputs and outputs (e.g., port numbers) need to be known.

Low cost. DiverseAV must achieve all the above properties with minimal computational and area cost overhead.

[3]Dynamic instances of an opcode are the actual instructions of that opcode that are fetched and executed by the processor.

B. Design Principles

In this work, we propose and describe our implementation of DiverseAV, which exploits the principle of temporal data diversity and redundancy. DiverseAV is an innovative redundant design for autonomous vehicles consisting of two independent software agents that are dynamic instances of the same underlying agent models (software) and are *time-multiplexed* on the shared computational fabric to actuate the vehicle. Time-multiplexing allows the following:

Semantic consistency. Each agent consumes semantically similar data. The sensing data used by the time-multiplexed redundant agents are semantically similar because the sensing frequency is typically very high, ranging from 10 Hz to 100 Hz among different sensors, and the world view (world semantics) does not change significantly between subsequent time steps.

Temporal data diversity. Temporal data diversity is enforced between the agents at the bit level (bit-level diversity). Sensor data obtained at consecutive time steps are semantically similar but differ significantly at the bit level. For example, objects captured in the camera frame at time t continue to exist in the frame at time t+1 with small shifts in their positions. However, at a pixel location, temporal data diversity is enforced because the pixel values and hence the bits representing that pixel can change significantly between subsequent frames due to shifts in object locations.

Error detection via time-multiplexing. Time-multiplexing enables detection of hardware errors that propagate from hardware to the software state and subsequently impact the AV dynamics and safety. Time-multiplexing of the sensor data between the DiverseAV-enabled agents detects a wide range of SDCs, since faults either impact a single agent or impact individual agents differently. A fault manifestation in each agent may be different because each agent consumes diverse (though semantically similar) data inputs and maintains its own private state.

C. Model formulation

Here we present the mathematical abstraction of our system. An ADS can be abstracted using (1). u_t is the actuation output produced by autonomous software (expressed as f) at time t using sensor data input I_t on a processing element (expressed as h). Let f^0 be the instantiation of f.

$$u_t = h(f^0, I_t) \tag{1}$$

Given an ADS that is well designed and trained and receives semantically similar data over a small rolling window of size rw, the average difference between adjacent actuation values over the rolling window of size rw are small and bounded, i.e., $\sum_{t=k}^{t=k+rw} ||u_{t+1} - u_t|| / rw \leq \delta$ as seen in Fig. 2(3)(a). It is plausible to develop a monitor to find anomalies in the timeseries data measuring δ to detect errors but such a monitor

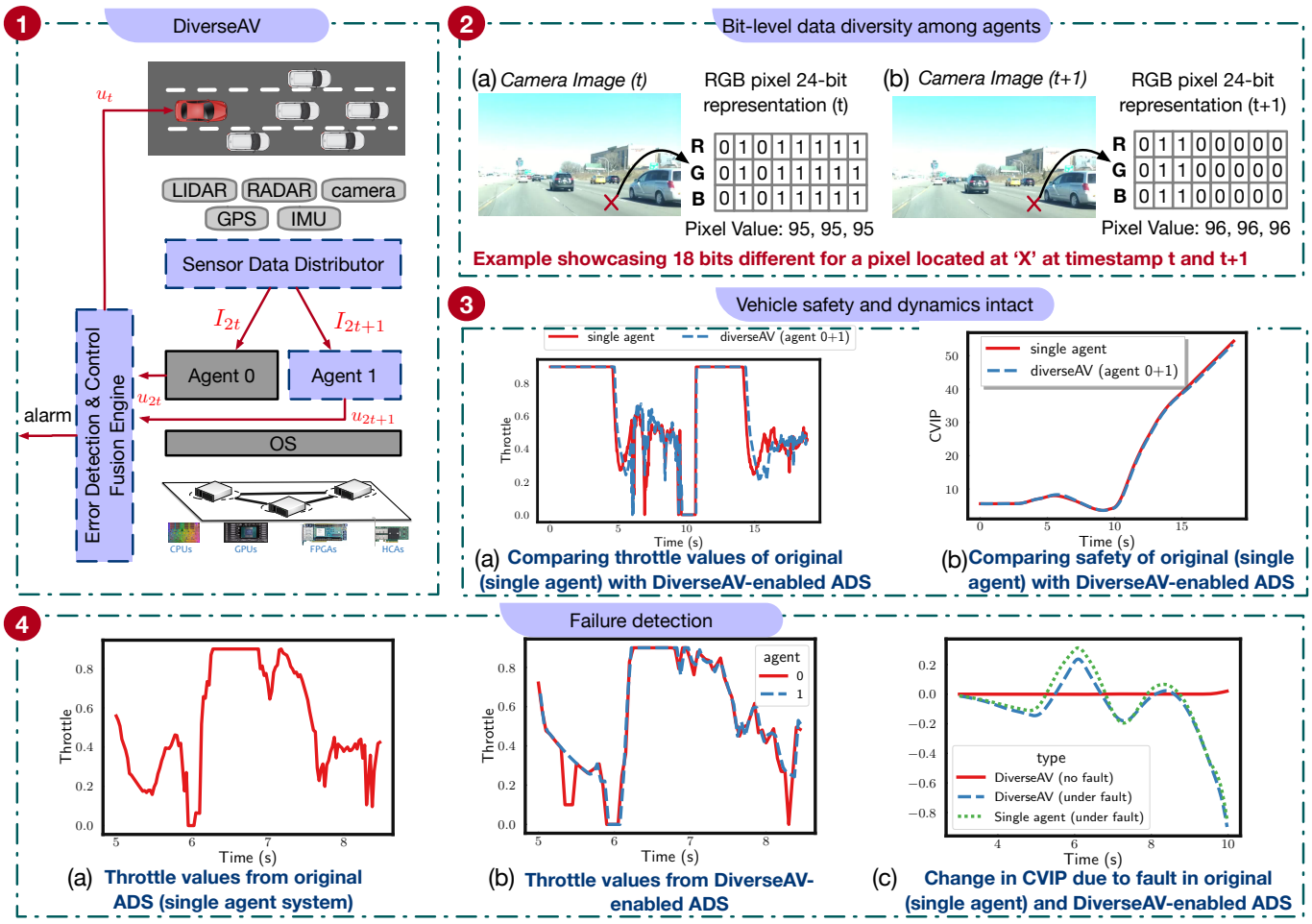


Figure 2: DiverseAV approach overview.

is noisy leading to high false positive rates as discussed in §VI-C.

In comparison, DiverseAV-enabled ADS can be modeled by (2), in which the data is distributed to the agents 0 and 1, each of which execute function f, in round-robin fashion using a 'sensor data distributor'. Here, $\mathbb{1}_{t=2k}$ and $\mathbb{1}_{t=2k+1}$, where k is a non-negative integer ($k \in \mathbb{Z}_{\geq 0}$), are indicator functions that activates on even ($t = 2k$) and odd ($t = 2k+1$) time steps respectively.

$$u_t = \mathbb{1}_{t=2k} h(f^0, I_t) + \mathbb{1}_{t=2k+1} h(f^1, I_t), k \in \mathbb{Z}_{\geq 0} \quad (2)$$

(1) and (2) are equivalent iff the composite function $h(f)$ is stateless, but, in practice, $h(f)$ is not stateless. However, (2) approximates (1) when the operating frequency of the ADS tends to infinity. This is because semantically I_t and I_{t+1} are similar even though I_t and I_{t+1} are not similar at the bit-representation level, i.e., $||w(I_{t+1}) - w(I_t)|| \to \epsilon$, where w is a function that extracts the semantic meaning from the image (e.g., bounding box of the objects or position of the object in the world), and ϵ is bounded and small. This is a fair assumption because practical implementations of ADSes operate at high frequencies (10-100Hz). However, this assumption is violated when the hardware is faulty. Under faulty hardware (expressed as h^τ), a DiverseAV-enabled ADS can be represented by (3).

$$u_t^\tau = \mathbb{1}_{t=2k} h^\tau(f^0, I_t) + \mathbb{1}_{t=2k+1} h^\tau(f^1, I_t), k \in \mathbb{Z}_{\geq 0} \quad (3)$$

Previous research [23], [35] as well as our own empirical demonstration of DiverseAV have shown that a faulty hardware (h^τ) executing an application produces significantly different outputs even when using semantically similar inputs when the input data is diverse. In our case, the input data is diverse at the bit level which is quantified in §V-A. Because of this diversity, the average error between adjacent actuation outputs produced by the two agents over a rolling window of size rw is neither small nor bounded, i.e., $\sum_{t=k}^{t=k+rw} ||u_{t+1}^\tau - u_t^\tau|| / rw > \delta$; thereby, enabling us to detect the error using a statistics-based 'Error Detection' engine.

D. Design Overview & Implementation

Fig. 2 shows the overall design (and envisioned deployment) of DiverseAV (1). The modifications to the original ADS system are highlighted in boxes with dashed blue outlines. To enable time-multiplexing between the agents, we introduce a "sensor data distributor" and an "error detection and control fusion engine."

Sensor data distributor takes the sensor data as inputs (I_t) and round-robins the input data among the two agents, thereby reducing the sensing frequency for each agent by 50%. For example, it splits the input data I_t such that agent 0 receives the input data $I_{t=2k}$ and agent 1 receives the input data $I_{t=2k+1}$, where k is a non-negative integer ($k \in \mathbb{Z}_{\geq 0}$). This decrease in sensing frequency of each agent by half is disadvantageous as it may have a negative consequences for AV safety because it reduces the accuracy of available history for making the driving decision, which may lead to an increase in uncertainty

or delayed response (e.g., uncertainty in a planner response). However, in [24], the commercial ADS was shown to be safe even with a sensing frequency that is much lower than the nominal rate because commercial ADS's include a significant engineering margin[4]. For example, across all driving scenarios, authors were able to decrease the sensing frequency by 3×, i.e., from 30 Hz to 10 Hz, safely.

Given the safety margin, DiverseAV can afford the data distribution strategy, which has several advantages: it ensures that each agent uses semantically similar sensor data to compute the actuation decision while providing significant data diversity at the bit level for the two agents. As can be seen from Fig. 2 (2), the camera frames captured at two consecutive time steps t and $t+1$ are semantically very similar; however, when they are compared at the bit level, their data are significantly different. For example, when the 24-bit RGB color value (8-bit per color) for a given pixel at location X changes from 95 (for each color at time t) to 96 (at time $t+1$), the data at the bit-level changes by 18 bits. We evaluate this temporal data diversity in detail in §V-A and show that the median number of bit difference per pixel location between successive camera frames is 8 bits. Thus, the sensor data distributor provides the much-needed data diversity to enable error detection. However, it also introduces several complications, such as ones related to synchronization and selection of the actuation decisions produced by each agent.

Control fusion engine is responsible for synchronization of actuation decisions between the two agents. Recall from §II that ADS fuses the sensor data spatially and temporally to produce actuation decisions and drive the vehicle in the real world. Depending on the ADS design, sensing and actuation can be (i) a lockstep process (i.e., an actuation decision is produced only after all inputs have been received, leading to the same sensing and actuating frequency as the original single agent system), as in the case of the Sensorimotor agent (described later in §IV); or (ii) an asynchronous process, as in the case of Baidu's Apollo agent [36]. The Apollo agent uses an array of sensors that post data at different frequencies (20 Hz cameras, 50 Hz radars, 100 Hz GPS and IMU (Inertial Measurement Unit), and 10 Hz LiDAR) to create an internal model of the real world and continuously update the internal world model. The planning and actuation model asynchronously uses the internal world model to produce the actuation decision at 100 Hz. Implementation of DiverseAV for the above-mentioned lockstep design is straightforward: DiverseAV can use the actuation decision of the agent that received the sensor data. However, implementing DiverseAV for an asynchronous design can be challenging: with two agents, DiverseAV doubles the number of actuation decisions produced by the ADS. Furthermore, enforcing an ordering of the actuation decisions across the agents is not trivial. Therefore, for an asynchronous system, DiverseAV can either (i) use an actuation decision from only one of the agents and use the actuation decision of the other agent solely for the purposes of error detection, or (ii) use the actuation decisions of both agents by averaging the actuation decisions produced by the replicas.

Fig. 2(3) shows the vehicle "throttle" actuation command value and CVIP distance ("closest-vehicle-in-path," described in §II) for the original system when it is using a single agent and DiverseAV-enabled ADS for the lead-slowdown driving scenario in which the lead vehicle is slowing down. Although the actuation decisions produced by DiverseAV-enabled ADS diverge from those of the original ADS (i.e., the single agent system) by a small amount, the CVIP distance shows negligible divergence. These results are described in §V-B in more detail.

Error detection engine. In a fully duplicated system since the agents are consuming the same input data the outputs can be compared directly using algebraic subtraction [5]. An alarm is raised if the subtraction yields a nonzero value. However, such systems are hard to design and implement. Designing such a system requires synchronization of the outputs at the instruction level, which is prohibitively costly. In contrast, our time-multiplexed redundant design leverages data diversity to detect safety-critical faults with marginal decrease error detection coverage while increasing the system availability, lowering the design complexity and significantly reducing the resource overhead compared to the fully duplicated systems. However, designing the error detection logic for DiverseAV becomes challenging as the diversity in the inputs and internal software state of the two agents leads to outputs that are not a bit-by-bit match. Thus, the challenge is to design a robust error detector that provides high detection accuracy, coverage and lead detection time. The *detection accuracy* is measured in terms of precision ($\frac{\#\text{True Positives}}{\#\text{True Positives} + \#\text{False Positives}}$) and recall ($\frac{\#\text{True Positives}}{\#\text{Positives}}$). The *lead detection time* is the difference between the alarm generation time and the collision time. An error detector with a higher lead detection time allows the ADS to switch over to fail-safe mode earlier.

We use statistical techniques to address the above-mentioned error detection challenge in DiverseAV. Fig. 2(4) depicts the impact of a permanent GPU fault on the original ADS and DiverseAV-enabled ADS for the lead-slowdown driving scenario. One can observe that the throttle values are different in the faulty run (Fig. 2(4)(a)) and a non-faulty run (shown in Fig. 2(3)(a)). Since the impact of the fault is smoothed by the PID controller, there are no visible anomalies in the throttle values for the original single-agent system (Fig. 2(4)(a)). However, one can see visible divergence between the outputs of the two agents in the DiverseAV-enabled ADS (Fig. 2(4)(b)).

Training error detection engine. DiverseAV uses a rolling window-based error detection algorithm to learn the maximum divergence between the actuation outputs of the two agents, and uses that divergence as a threshold to detect errors at runtime. We ensure that DiverseAV is not tuned to any specific scenarios or faults by training the error detection engine (i) using the *long training scenarios*, described in §IV, which is different from our evaluation scenarios (e.g., it does not include emergency braking or accidents), and (ii) by executing these scenarios under fault-free conditions.

At runtime, DiverseAV uses the learned divergence parameters to detect a safety-critical fault. Upon detection of a safety-critical fault, an alarm is raised, and DiverseAV triggers a fail-back system with sufficient capabilities to handle the driving situation, e.g., safely park the vehicle. The rolling window-

[4]For an ADS with lower engineering margins, the sensor data distribution can be adjusted so that some input data is sent to both agents, thus resulting in a input data rate reduction less than 50%, albeit at the expense of greater performance overhead.

[5]Note that depending on the granularity at which a fully duplicated system is synchronized the outputs may or may not be directly comparable. Here we assume that the duplicated system is synchronized at the processor instruction level. Later in §VI-B we discuss other synchronization granularities.

based error detector uses the following parameters:

(i) $\theta_{throttle}(s), \theta_{brake}(s)$, and $\theta_{steer}(s)$: DiverseAV raises an alarm if the difference between the actuation command values of the two agents exceeds a certain threshold at a given vehicle state s (given by tuple $\langle v, a, \omega, \alpha \rangle$, where v is speed, a is acceleration, ω is angular velocity, and α is angular acceleration). We use $\langle v, a \rangle$ to represent the state for $\theta_{throttle}(s), \theta_{brake}(s)$, since the throttle and brake depend on linear speed and acceleration. Similarly, we use $\langle \omega, \alpha \rangle$ to represent the state for $\theta_{steer}(s)$.
We discretize each of the variables (i.e., $\langle v, a, \omega, \alpha \rangle$) of the vehicle state s into small intervals and learn the thresholds for each of these intervals. The thresholds are learned by calculating the maximum difference between the actuation command values for the two agents across all executions of reference driving scenarios for a given vehicle state (s). The thresholds learned are stored in a lookup table (LUT), which is used at runtime for detection.

(ii) rw: The two agents in DiverseAV naturally produce slightly different actuation command values because they are consuming diverse data, and the divergence is highest when the planning decision changes between two time steps (e.g., from slowing down to accelerating). However, such high divergence in actuation is transient. Therefore, to avoid identifying occasional blips as errors, we use a rolling window (rw–rolling window size) to smooth out the difference in actuation command values produced by the two agents. The rw parameter may impact the lead detection time. We vary the rolling window from 3 all the way to 40 recently received sensor data, as 40 Hz is the sensor frequency of our simulator, and select the parameters with maximum F1-score (harmonic mean of precision and recall).

IV. Experimental Setup

This section describes the autonomous agent, simulation platform, driving scenarios, data collection methods, and fault injection methods used in our experiments. Hereafter, we refer to the agent-controlled vehicle as “the ego vehicle" and the other vehicle in a scenario as “NPC (non-player character)".

A. Autonomous Agent

This work uses the state-of-the-art convolutional neural network (CNN)-based end-to-end autonomous agent proposed and pretrained by Chen et al. [26], referred to as the *Sensorimotor agent*. The main components of the agent are the High-level Route Planner, CNN, Waypoints Tracker, and Control Unit. *High-level Route Planner* is responsible for finding the next “destination-to-go” navigation direction. *Convolutional Neural Network (CNN)* is a vision-based local planner, and is the core of the Sensorimotor agent. The CNN consumes data from three front-facing cameras and predicts the path that the ego vehicle should follow by outputting four local-waypoints for each time step. *Waypoints Tracker* along with the *Control Unit* uses the local waypoints, the IMU and GPS data, and a PID controller to produce actuation outputs at each time step.

B. Simulation Platform

We cannot use real-world data (such as KITTI dataset [37], [38]) to evaluate DiverseAV as the fault may impact the future ego vehicle trajectory, and therefore, the subsequent data captured via the sensors. Thus, in this work, we use a world-simulator to simulate the driving scenarios. An overview of our

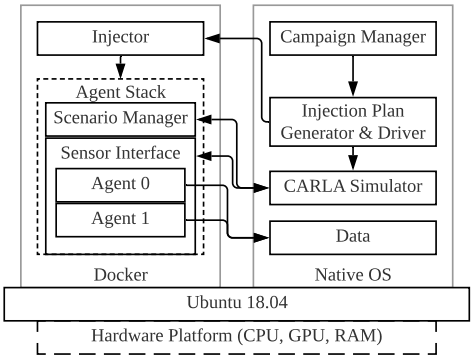


Figure 3: DiverseAV assessment platform.

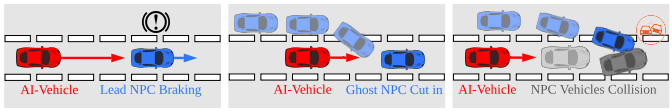


Figure 4: Driving scenarios. Left: lead slowdown. Middle: ghost cut in. Right: front accident. Red car: AI-vehicle, Blue Car: NPC-vehicle.

world-simulation platform is shown in Fig. 3. We use CARLA 0.9.10 [27], an Unreal Engine-based simulator, to simulate complex and realistic 3D environments for autonomous driving. We ran the CARLA simulator in synchronous mode with all sensor data (from 3 front-facing cameras (facing left, center, and right) and GPS and IMU) posted at 40 Hz.

The DiverseAV-enabled ADS consists of two Sensorimotor agents, a sensor interface for communication with the simulator, and a scenario manager that manages the driving scenario. The two agents can be configured to run in round-robin mode (i.e., agents receive sensor data at alternating time steps) , duplicate mode (i.e., both agents receive all sensor data), or single mode, in which only agent 0 is active.

Before the simulation is started, the Scenario Manager sends the driving scenario to the CARLA simulator. The Campaign Manager reads experiment configurations and launches the Injection Plan Generator that selects the injection site (CPU vs. GPU), the fault model (transient vs. permanent) and agent mode (single, duplicated, or DiverseAV) for an experiment. The Driver invokes the simulator with the selected agent mode and the selected fault injector. The Campaign Manager takes approximately 10 minutes to run one experiment which includes setting up the agents, the simulator, executing the test driving scenario, selecting and injecting faults, and collecting the data.

C. Driving Scenarios

1) Safety-critical (Test) Scenarios: We created three safety-critical test scenarios, shown in Fig. 4. Scenarios of these kinds are considered high-risk by the National Highway Traffic Safety Administration (NHTSA), as stated in their pre-collision scenario topology report [39]. The safety-critical scenarios are about 30-60 seconds long on wall clock[6] and capture the most critical moments of autonomous driving. We used these safety-critical scenarios in fault injection experiments to evaluate the the efficacy of detecting safety-critical faults.

Lead Slowdown: As shown in Fig. 4 (left), the ego vehicle (red) follows a leading NPC vehicle (blue), maintaining a distance of 25 meters. The NPC vehicle then performs emergency braking to slow down. Lead slowdown scenario is

[6]The simulation time corresponding to 30-60 seconds of driving is as high as 5-10 minutes on the wall clock time.

dangerous because it gives the follower vehicle little time to react, often resulting in a rear-end collision with the leading vehicle.

Ghost Cut in: As shown in Fig. 4 (middle), the ego vehicle (red) is maintains a constant speed, and an NPC vehicle (blue) approaches from the left adjacent lane. The NPC vehicle then cuts in front of the ego vehicle with a small longitudinal margin. The ego vehicle needs to reduce the throttle, slow down, and brake if necessary to avoid colliding with the side of the NPC vehicle. In this driving scenario, there is little to no warning prior to the cut-in maneuver of the NPC vehicle. This is especially dangerous for the ego vehicle as our agent does not use a rear-end camera, resulting in less time to see the NPC vehicle and react to avoid a collision.

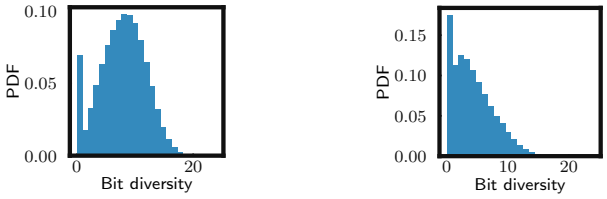
Front Accident: As shown in Fig. 4 (right), the ego vehicle (red) is following a leading NPC vehicle (gray) in the same lane, and another NPC vehicle (blue) in the adjacent lane tries to merge but crashes into the leading NPC vehicle. Both NPC vehicles' trajectories suddenly change because of their collision, and both NPC vehicles stop subsequently. Although a scenario with accident is rare, it is a high-risk situation because the ego vehicle might not recognize the abrupt change in the perspective and the trajectory of the leading NPC vehicle and makes the wrong decision.

2) Long (Training) Scenarios: We constructed three long scenarios for training the error detector of the DiverseAV-enabled ADS. Our results, described in §V, show that the error detector parameters can be learned from these long driving scenarios with high precision and recall in detecting safety-critical faults. The long scenarios are based on selected routes from the 2020 CARLA Autonomous Driving Challenge [40], simulating normal, everyday driving tasks, such as vehicle following, lane keeping, turning, lane changing, and handling of intersections. We also enabled pseudo-random background traffic (consistent across all experiments of each scenario) with a fixed random seed for each run. Each driving scenario simulation time is approximately 10–15 minutes long. The three long scenarios are based on Route02, Route15, and Route42, which are set in CARLA Town01, Town03, and Town06, respectively. These long scenarios require the AV to navigate in city and highway with dense traffic consisting of turns, intersections, and traffic lights.

D. Fault Injection

We inject hardware faults by injecting architectural-level GPU or CPU errors that emulate consequences of underlying transient or permanent faults as discussed in §II-A. In particular, we use `PinFI` [29], [41] to inject faults into the CPUs, and `NVBitFI` [28], [34] to inject faults into the GPUs. We configure the fault injection tools so that they only affect the ADS (the agents in Fig. 3).

GPU fault injections. We conduct the following GPU FI experiments. (i) *Transient FI*: It is prohibitively expensive to inject all possible transient faults as the possible space of transient faults is extremely large. Therefore, we uniformly randomly selected 500 candidate dynamic instructions to transiently corrupt the destination register, with one fault per simulation run. (ii) *Permanent FI*: For permanent faults, we inject all dynamic instances of the target opcode for a particular run. The ISA (Instruction Set Architecture) of the Titan Xp GPU includes 171 opcodes, and for each of the three driving scenarios we perform fault injection for all 171 opcodes, with three repeated runs per opcode to capture any non-deterministic effects, resulting in 513 experimental runs per driving scenario. In total, the fault injection campaigns on GPUs, which includes both transient and permanent fault injection on all scenarios, lasted for 21 days.



(a) Real-world data from the KITTI dataset.

(b) Simulated data from the CARLA simulator.

Figure 5: Image pixel bit diversity.

CPU fault injections. We conduct the following CPU FI experiments. (i) *Transient FI*: Similar to the GPU FI experiments, we uniformly randomly select 500 candidate dynamic instructions and transiently corrupt the destination register. (ii) *Permanent FI*: The Sensorimotor agent uses 131 Intel opcodes, and for each of the three driving scenarios we perform fault injection for all 131 opcodes, with three repeated runs per per opcode to capture any non-deterministic effects. We also perform injection of CPU faults using a modified version of `PinFI` to support a permanent fault model that is similar to the permanent fault model for `NVBitFI`, where all dynamic instances of a specified opcode are corrupted, resulting in 393 experimental runs per driving scenario. In total, the fault injection campaigns on CPUs, which includes both transient and permanent fault injection, lasted for 18.6 days.

In addition we run 50 experiments per scenario without fault as "golden" runs. The golden runs serve as control experiments as the error detector *must not* classify any of these runs as faulty. An error detector which falsely classifies a golden run as erroroneous will trigger frequent alarms, thereby decreasing the overall system availability. Finally, our experimental setup uses a XEON E5-2699v4 CPU with 64 GB of RAM and two Titan Xp GPU cards.

V. Results

A. Sensor Data Diversity and Semantic Consistency

We characterize the diversity in the sensor data between two consecutive time steps of autonomous driving on both the simulated sensor data generated by CARLA Simulator [27] for our test scenarios (§IV-C) and the KITTI dataset [37], [38]. The KITTI dataset is a real-world dataset for autonomous driving recorded in various scenarios representing real-world traffic from both urban and highway-driving with many static and dynamic objects. The KITTI dataset consists of data captured by multiple sensors and object labels, including two front-facing cameras, one Velodyne 64-channel LiDAR, one IMU+GPS sensor, and ~200k 3D object labels. The sensing frequency is 10 Hz for all sensor data.

The camera data diversity at the bit level is calculated per pixel by counting the number of bits that are different at a given pixel location between consecutive RGB camera images. Such a calculation is done for all pixel locations, resulting in a distribution. For KITTI dataset, the bit diversity of the camera data is 8 bits and 13 bits out of the 24-bit RGB pixel (8 bits

per channel) at the 50th and 90th percentile, respectively (Fig. 5a). Further evaluation shows that the bit diversity remains high for other sensor data. The bit diversity of the IMU+GPS data (using 32-bit floating points) is 11 bits and 15 bits at the 50th percentile and 90th percentile, respectively. The bit diversity of the LiDAR data (using 32-bit floating points) is 14 bits and 18 bits at the 50th and 90th percentile, respectively.

The above characterization on bit diversity holds for simulator-generated dataset also. We evaluate the bit diversity of CARLA simulator-generated sensor data captured from the three front facing cameras running at 40 Hz (the primary data consumed by Sensorimotor agent) on the test (safety-critical) scenarios. The bit diversity of simulator-generated camera data is 5 bits and 9 bits out of the 24-bit RGB at 50th and 90th percentile, respectively (Fig. 5b). We omit the detailed analysis for other simulator-generated sensor data due to lack of space.

Moreover, we estimate the semantic consistency of the sensor data of the KITTI data set between two consecutive time steps. For the camera data, we calculate the shift of the object center in pixel coordinates per object between two consecutive frames using the ground-truth 2D bounding box labels from KITTI's object tracking task [38]. Our results show that the pixel shift of the bounding box center between two consecutive frames is 5 and 22 pixels at the 50th and 90th percentile, respectively, with the maximum possible shift being ~1296 pixels (the diagonal of a 1240x376 KITTI camera frame [38]). For the LiDAR sensor data, we calculate the shift of the object center in the ego vehicle's coordinate frame per object between two consecutive frames using the ground-truth 3D object center label [38]. Our evaluation shows that the object's position difference in the ego vehicle coordinate is 0.48 and 1.26 meters at the 50th and 90th percentile, respectively, with the maximum possible shift being 240 meters (the LiDAR's effective range [37]).

These results validate our assumptions and show that even though the semantics meaning of the sensor data does not change significantly between two consecutive time steps, the bit-representation can change considerably.

B. Impact of DiverseAV on Safety

Here we characterize the impact of the DiverseAV-enabled ADS on the vehicle's safety by evaluating the maximum divergence between the trace of the vehicle trajectory ($traj$) of an experimental run of a driving scenario generated using the DiverseAV-enabled ADS and the baseline trajectory generated using the original single-agent ADS. The *vehicle trajectory* of an experimental run of a driving scenario is the trace of the path followed by the vehicle. Formally, it is a timestamped list containing the global position (in terms of <x,y,z> coordinates) of the vehicle at any time t on the map during the execution of that driving scenario, i.e., $traj = [pos_t | \forall t]$. We express the maximum divergence between a given trajectory ($traj^E$) and the baseline trajectory ($traj^B$) as $\delta_{pos}^{E,B}$, where $\delta_{pos}^{E,B} = max(traj^E - traj^B)$.

Fig. 6 shows the boxplot of $\delta_{pos}^{E,B}$ across three safety-critical driving scenarios, calculated using 50 experimental runs (golden runs) of the scenarios. We characterize the divergence among the vehicle trajectories generated using the original ADS as well as the DiverseAV-enabled ADS. The baseline trajectory $traj^B$ used for calculating $\delta_{pos}^{E,B}$ for a given driving scenario was chosen as the mean of all the golden trajectories generated using the original ADS. Thus, the boxplots labeled "orig" show

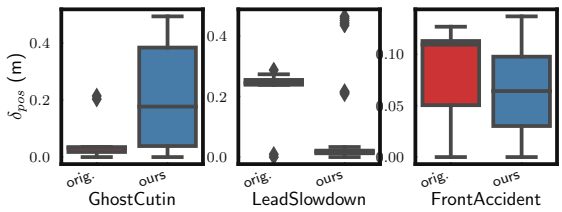


Figure 6: Impact on vehicle trajectory for a single instance (orig) and with DiverseAV (ours).

the distribution of the maximum divergence for the vehicle position across the experimental runs of a driving scenario when the vehicle was using the original ADS. Similarly, the boxplots labeled "ours" show the maximum divergence for the vehicle position among experimental runs when the vehicle was using the DiverseAV-enabled ADS with respect to the mean of the trajectories generated by the original ADS. Our characterization shows that the maximum divergence between the vehicle trajectories was <50 cm across all scenarios when we used the DiverseAV-enabled system instead of the original ADS. Note that the divergence was mostly observed longitudinally rather than laterally. Hence, the maximum observed divergence is significantly less than the distance that the vehicles must maintain with other actors (as dictated by the law) to avoid accidents. For example, drivers should follow 3-second rule to always try to maintain a 3-second following distance whenever possible; providing sufficient space for emergency braking [42].

Moreover, the DiverseAV-enabled vehicle neither experienced a collision nor broke any traffic laws in any of our experimental runs across the driving scenarios. Based on those observation, we conclude that our proposed design is safe and mimics the vehicle trajectory of the original ADS closely.

C. Characterizing Fault Propagation

Table I provides an overall summary of fault injection experiments. Each row in the table shows the statistics for a fault injection (FI) campaign, which is characterized by a fault injection target and the driving scenario. In total, we executed twelve FI campaigns in which we injected faults into two targets (CPU and GPU) in three driving scenarios (LeadSlowDown, GhostCutin, and FrontAccident). We also used three additional training driving scenarios (Town01-Route02, Town03-Route15, and Town06-Route46) to train our error detector (not mentioned in the table). For each of the campaigns, we ran 50 golden runs (i.e., experimental runs without fault injections) to (i) characterize the simulation's non-determinism, (ii) understand the impact of faults on the vehicle's safety, and (iii) test the error detector. Across all the FI campaigns, we quantify the safety of the vehicle in terms of accidents and trajectory violations. We marked an experimental run (E) as "trajectory violated" if $\delta_{pos}^{E,B} \geq td$ (i.e., the maximum divergence between the trajectory of the experimental run (E) and the baseline run (B) is more than td meters, where td is a parameter). We evaluate the sensitivity of the parameter td on the detection capabilities of our proposed design. The trajectory of the baseline run is calculated as the mean trajectory of all the golden runs. Note that the "baseline" trajectories are generated by taking the means of the golden runs of the original ADS in §V-B, while in this subsection and §V-D, the "baseline" trajectories are generated by taking the mean of the golden runs of DiverseAV-enabled ADS under fault-free condition.

FI Target	DS	#Active, Hang/Crash, Total FI	#Acc.	#Traj. Violations*
GPU-permanent	LSD	513, 83, 513	3	9
GPU-permanent	GC	513, 83, 513	14	2
GPU-permanent	FA	513, 81, 513	0	3
CPU-permanent	LSD	393, 287, 393	0	0
CPU-permanent	GC	393, 286, 393	0	0
CPU-permanent	FA	393, 287, 393	0	0
GPU-transient	LSD	500, 40, 500	0	2
GPU-transient	GC	500, 46, 500	0	2
GPU-transient	FA	500, 39, 500	2	0
CPU-transient	LSD	413, 171, 500	0	0
CPU-transient	GC	203, 70, 500	0	0
CPU-transient	FA	452, 199, 500	0	0

Table I: Summary of experimental runs in DUAL agent mode. DS: Driving scenarios (LSD - Lead Slowdown, GC - Ghost Cut in, FA - Front Accident scenarios); #active: # of FI experiments in which the injected fault has been activated, e.g., a corrupted data has been used; #Traj Violations*: # of experiments with trajectory violation but without accident. Here we assume that the trajectory violation occurs when the maximum divergence between the experimental run of a driving scenario with FI enabled and the baseline trajectory exceeds 2m. #Acc.: # of experiments with accident.

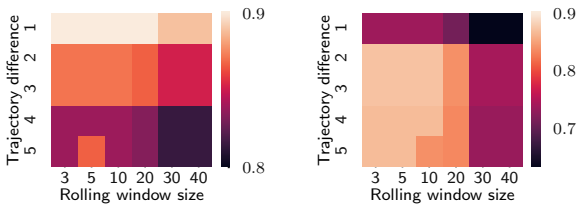
Transient faults. Across all transient faults, CPU FI resulted in (i) highest percentage of hangs and crashes of the ADS software (41.2%; 440 out of 1068 runs[7]), and (ii) zero accidents and trajectory violations. A high percentage of hangs and crashes are expected for CPU FI campaigns because FI into CPU instructions is very likely to corrupt the program control flow or memory addresses, resulting in segmentation faults and broken pipes, among other problems. Hangs and crashes are automatically detected by the platform, thereby triggering the fail-back system which can bring the vehicle to a safe state. CPU FIs do not cause silent data corruption (SDC) because the Sensorimotor agent used in our work uses the GPU mostly for computations, whereas it uses the CPU for loading and setting the `Pytorch` program. Consequently, we observed a relatively low percentage of hangs and crashes for GPU transient faults (8.3%; 125 of 1500 runs). Transient faults into GPU did lead to accidents and trajectory violations (0.4%; 6 out 1500 runs).

Permanent faults. We observe similar trends for permanent faults for CPUs and GPUs except for the fact that permanent faults resulted in significantly more hangs/crashes and accidents/trajectory violations. CPU FI resulted in (i) the highest percentage of hangs and crashes (72.9%; 860 out of 1,179 runs), and (ii) zero trajectory violations and accidents. Similar to transient FI, we observed a relatively low percentage of hangs and crashes (16%; 247 out of 1,539 runs) in the case of GPU FI as compared with CPU FI, and a high percentage of accidents (1.1%; 17 out of 1,539 runs) and trajectory violations (with no accident) (0.9%; 14 out of 1,539 runs).

D. Characterizing Error Detection Capabilities

DiverseAV must detect all safety-critical errors, i.e., faults that lead to a collision or significant trajectory divergence. In addition, it should not raise false alarms, especially for the golden runs (experimental runs of driving scenarios without fault injection). We evaluate error detection capabilities in terms of precision, recall, and lead detection time. Moreover, we parameterize the trajectory divergence using the parameter td. We mark an experiment as "trajectory violated" if $\delta_{pos}^{E,B} \geq td$, i.e., if the max difference between the experimental run of a driving scenario and the baseline trajectory exceeds td. This parameter impacts the number of cases that need to be detected by the DiverseAV. We evaluated DiverseAV's detection capabilities for $td = 1, 2, 3, 4, 5$ meters. The simulations of the driving scenarios have inbuilt non-determinism, and, as shown in Fig. 6, the natural variation in trajectory can be as high as 50 cm; therefore, we chose $td > 50\ cm$.

[7]The statistic is calculated by dividing the total number of experiments with hangs and crashes due to activated CPU-transient faults by the total number of experiments with activated CPU-transient faults (see Table I column 3).



(a) Precision. **(b)** Recall.

Figure 7: Detecting safety-critical GPU faults.

We omit the results for CPU faults as all the CPU faults were either detected by platform (i.e., OS or Scenario Manager) as hangs or crashes or did not result in accidents or trajectory violations. Since the undetected faults by the platform did not result in any safety violation, we institute a simple policy in which DiverseAV generates an alarm if it detects a hang or crash. Because we did not observe any SDCs, we cannot estimate the efficacy of DiverseAV in detecting SDCs. However, we expect the contribution of CPU faults to the total SDC FIT rate to be small.

Going forward we only discuss the error detection capabilities of DiverseAV for GPU faults as undetected GPU faults lead to safety violations. We trained and tested DiverseAV on different scenarios to understand the generality of the proposed design. DiverseAV was trained using "long driving" scenarios and tested on safety-critical scenarios. Fig. 7a and Fig. 7b show heat maps of the precision and recall values for our error detector across different parameters of td and rw (rolling window size). Overall, we found that the DiverseAV's detector robustly detected the safety-critical faults and produced a low false positives rate across a range of parameters ($td \geq 2m$ and $rw \leq 30$). The best performance (i.e., precision = 0.87 and recall = 0.87) was achieved with $td = 2m$ and $rw = 3$. Thus, DiverseAV generates few false positive alarms and detects majority of the safety violation causing faults; thereby improving the overall safety of the system. Moreover, DiverseAV did not raise an alarm for any of the golden runs of the driving scenarios for these parameters, which means the false positives are cases in which a fault did occur but did not result in any safety violation (i.e., accident or $td > 2m$). Fig. 8 shows the lead detection time for the detector using the parameters $td = 2m$ and $rw = 3$. The lead detection time is significantly higher than 1.0 second. [5], [43] found that the reaction time of braking for humans and AVs to be 0.82 second and 0.85 second, respectively. Therefore, DiverseAV provides sufficient time to take control and react to the driving situation at hand for humans or a fail-back system.

The above results indicate that our approach achieves high recall and precision in detecting runtime errors. In the context

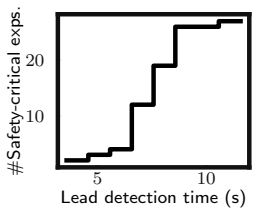


Figure 8: GPU FI lead detection time. In this plot, at any point (x,y), y is the total number of safety-critical experiments in which the lead detection time is $<= x$ seconds.

	CPU	GPU	RAM	VRAM
Single Agent	4%	14%	431 MB	198 MB
DiverseAV	5%	15%	862 MB	396 MB
FD*	4%	14%	862 MB	396 MB

Table II: Average system resources used by single-agent, DiverseAV-enabled and fully duplicated (FD) ADS. $*$: CPU and GPU utilization are per processor for FD.

of an ADS, high recall and a corresponding low false-negative rate are important because they indicate that most faults are being detected or they do not affect the vehicle behavior. When an error is detected, the vehicle fails over to a backup system that brings the vehicle to the safe state.

E. Performance Overhead

A summary of performance overhead is shown in Table II. The performance overhead characterization shows that DiverseAV increased compute utilization marginally and significantly increased (by $2\times$) the memory utilization. That is expected because of the two agents employed in DiverseAV maintain their own internal (private) state. Memory usage (for both CPU and GPU) doubled as expected for DiverseAV compared to the single-agent system and was the same compared to the fully duplicated (FD) system. DiverseAV used slightly more compute resources than the single-agent system. Compared to the FD system, DiverseAV averaged the same compute utilization on a per-processor basis. However, the FD system requires double the number of CPUs and GPUs.

Although the compute resource utilization is low for the Sensorimotor agent used in this paper, we know from our experience that compute utilization for a real-world AV is high, requiring multiple CPUs, GPUs, and FPGAs [44].

VI. Discussion

A. Assumption of independence of agents and its impact on error detection coverage.

A key assumption of DiverseAV is the implementation of the two agents as separate processes executing on the same processor. Thus, an error in one agent cannot directly affect the other. An error could affect the operating system, which in turn could affect both agents, but such errors would most likely result in either crashes or hangs. Hence, the assumption is that transient hardware faults can only affect a single agent. Permanent faults would affect both agents, but our evaluation shows that the diverse data state of the two agents results in actuator commands that diverge sufficiently to trigger an error detection. However, there is a nonzero chance that the diverse agents will produce similar actuation outputs even in the presence of an error. In our experiments, we find the probability that a fault will result in similar actuation outputs in both agents and also result in a safety hazard to be small (0.001 for GPU faults; which is estimated by calculating $^{\text{missed safety hazard cases}}/_{\text{total fault injection experiments}} = {}^{4}/_{3189}$).

B. Comparison with Fully Duplicated ADS

Full duplication of an ADS can be achieved at different levels of granularity, ranging from at the instruction level (tight coupling) to the vehicle actuation level (loose coupling). By controlling the granularity, the designer explicitly trades off error detection coverage and system availability for design and implementation simplicity. We implemented and evaluated a loosely coupled duplicated system, which is hereafter referred to as FD-ADS. We only implement and evaluate FD-ADS and not the tightly coupled duplicated system because: (i) synchronization at the instruction level, which is required for tightly coupled systems, is prohibitively expensive and difficult to achieve in practice and (ii) tightly coupled systems will detect all faults irrespective of their plausible masking at the actuation level by the application, which significantly decreases the availability of the system.

In FD-ADS, the two redundant agents are executing on their own dedicated processors receiving and processing the same input data from the same set of sensors (i.e., sensors are not duplicated). We do not multiplex the duplicated agents on the same processor because (i) the resource overhead of executing the two agents doubles at the original sensing frequency, and (ii) multiplexing on the same hardware will not detect permanent faults as both the duplicated agents will produce the same output as they receive the same input and experience exactly the same fault. We inject a fault in one of the agents while using the other one as a reference for comparison. Because the trace of the agents' outputs do not match bit-for-bit (even for the golden runs) due to the inherent non-determinism that exists in software and hardware, we use a statistical model to detect errors. The error detector is trained using the rolling window-based approach discussed in §III.

Compared to DiverseAV, FD-ADS achieved a precision of 0.18 and a recall of 0.84 across 500 runs of each scenario (1500 total runs). FD-ADS correctly identified most cases of true positives (accidents and trajectory violations). However, it did not result in recall of 1.0 because we use statistical error detection as there is inherent non-determinism in software and hardware when the agents are synchronized at the actuation level. A tightly coupled duplicated system would have reached a recall rate of 1.0; however, as discussed earlier it is prohibitively expensive to design and implement such a system. Note that the recall values of DiverseAV and FD-ADS are close to each other, showcasing the efficacy of DiverseAV. Not surprisingly, FD-ADS falsely identified significant number of fault-injected runs which did not lead to safety hazards as errors. This is because it is overly sensitive to mismatches between the control outputs even when they would not lead to safety hazards, leading to lower precision (and lower availability) compared to DiverseAV-enabled ADS. Similar to the DiverseAV-enabled ADS, none of the golden runs were marked in error.

C. Comparison with Single Agent ADS

We compared our model with the single-agent system, in which the ADS is using *only* a single agent to control the ego vehicle. Both in the FD-ADS and DiverseAV-enabled ADS, the system is using two agents and hence, the agents are reference

to each other for comparing the outputs. However, in the single-agent system, there is no reference available for comparison except for identifying anomalies in the timeseries data.

It is difficult to identify errors using temporal outliers or range-based detectors [45] as occasional blips (caused by mode changes, such as from throttling to braking, that are within the acceptable output range) frequently occur in the actuation outputs (see Fig. 2(4)(a)). Increasing the rolling window size to smooth the outputs in order to remove blips reduces the overall recall of the error detector model, while decreasing the rolling window size results in too many false positives. To illustrate the difficulty in designing a temporal outlier-based error detector using a single agent, we developed a rolling window-based anomaly detection technique similar to the one used in this paper but use the agent's output from the previous time step as a reference. The best performance, in terms of detection accuracy, achieved by the single-agent system yields in precision and recall of 0.17 and 0.52 respectively; which is significantly lower compared to both FD-ADS and DiverseAV-enabled ADS. We must note that it might be possible to detect safety-critical faults in a single-agent ADS; however, that approach will require large amount of data and complex machine-learning models such as an LSTM/RNN [46] to train the detector. Our future work will explore such models. In contrast, DiverseAV is simple requiring comparison of actuation commands with interpretable statistical model using fewer model parameters.

VII. Related Work

Safety-critical systems employ one or more of the following techniques to protect against random hardware faults.

Hardware design modifications, which include error detection and correction at the circuit, micro-architecture, and architecture levels (e.g., instruction retry [47]–[49], ECC [50]–[52], checkers [53], and parity codes [8]). Significant effort has been devoted to hardware-level redundancy such as lockstep duplication [13]–[15], thread redundancy inside a single core [14], [16], or across cores [17]–[20], including partial redundancy techniques [54], [55]. However, those solutions require hardware support for thread synchronization, and incur significant area and power overheads. Furthermore, because of those overheads, applications of these techniques tend to be associated with larger arrays of circuit elements for which the overheads can be amortized. Thus, in typical chips, a significant portion of vulnerable elements are not protected (e.g., small SRAM arrays, flip-flops, compute units, and pipelines) leading to silent-data corruptions [56], [57]. Moreover, these techniques though largely effective do not provide full coverage, leading to escape of faults at the hardware level. Thus, there is a need for high-level software-based approaches to detect those escaped faults that are safety-critical at the software level.

Software algorithms or enhancements, which include error detection and correction at the software level with negligible dedicated hardware support. Techniques include (i) algorithm-based error detection [58], [59], (ii) assertions [11], [12] (iii) monitoring [60]–[63]), and (iv) software-based redundancy (e.g., instruction retry and duplication among others [64]–[69]). Software enhancements usually incur lower overhead than hardware methods. However, the applicability of software techniques tends to be dependent on the specific target software, so significant portions of the software are often left unprotected. Moreover, identifying algorithms and methods that provide high coverage is challenging and requires an in-depth understanding of the fault propagation in the application [70].

Enforcing diversity helps to tackle common cause failures (CCF) such as design bugs and software implementation defects. Diversity can be enforced at the instruction and program level [22], temporal level (e.g., instruction-retry), design level [21] and the data level [23]. The assumption is that the diverse designs are susceptible to different faults and therefore, the outputs of the two diverse designs will significantly differ on encountering a systematic fault. However, designing diversity is too costly (in terms of man-hours required to develop N-versions [22] or data transformation techniques/input reexpression logic [23]) and arguably challenging to enforce in practice.

Putting DiverseAV into perspective. DiverseAV is a lightweight, software-based redundancy technique that exploits the temporal data diversity present in the sensor data of dynamical autonomous systems to achieve high-coverage error detection for transient and permanent hardware faults without incurring significant computational overhead (in terms of performance and hardware/software resources), thus enabling detection of safety-critical faults in the computational hardware elements of the entire ADS. In contrast to full hardware or software duplication, DiverseAV ensures data and (internal) state diversity between the two agents. Moreover, DiverseAV is a plug and play solution, and the engineering and development effort for enabling DiverseAV is small (unlike for the above-mentioned diversity techniques). To the best of our knowledge, there is no existing work on achieving ADS redundancy by leveraging temporal data diversity in sensors.

VIII. Conclusion

This paper proposes and develops DiverseAV, a low-cost redundancy technique for autonomous driving agents that leverages temporal diversity for safety-critical error detection. Our results show that DiverseAV trades off a marginal decrease in error detection coverage to increase the overall system availability significantly and to lower design complexity and resource overheads compared to a tightly coupled fully duplicated system. In the future, we plan to address the limitations discussed in §VI. We also plan to explore the efficacy of DiverseAV in other dynamical systems such as unmanned aerial vehicles to understand its limitations, if any.

Acknowledgement

We thank Henrique Madeira (shepherd) and anonymous reviewers for insightful conversations and feedback. This material is based upon work supported by the National Science Foundation (NSF) under Grant No. CNS 15-45069. The first author was supported by the 2020 IBM PhD fellowship. We thank NVIDIA Corporation for software access and equipment donation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF, IBM and NVIDIA. We also thank J. Applequist and K. Atchley for proofreading this manuscript.

REFERENCES

[1] J. Tan and X. Fu, "Chapter 23 - addressing hardware reliability challenges in general-purpose gpus," in *Advances in GPU Research and Practice*, ser. Emerging Trends in Computer Science and Applied Computing, H. Sarbazi-Azad, Ed. Boston: Morgan Kaufmann, 2017, pp. 649–705. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128037386000239

[2] "Road vehicles — Functional safety," International Organization for Standardization, Geneva, CH, Standard, Dec. 2018.

[3] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding Error Propagation in Deep Learning Neural Network (DNN) Accelerators and Applications," in *Proc. International Conf. for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 8:1–8:12.

[4] ——, "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–12.

[5] S. S. Banerjee, S. Jha, J. Cyriac, Z. T. Kalbarczyk, and R. K. Iyer, "Hands off the wheel in autonomous vehicles?: A systems perspective on over a million miles of field data," in *Proc. 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018.

[6] S. Jha, S. Banerjee, T. Tsai, S. K. S. Hari, M. B. Sullivan, Z. T. Kalbarczyk, S. W. Keckler, and R. K. Iyer, "Ml-based fault injection for autonomous vehicles: A case for bayesian fault injection," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, June 2019, pp. 112–124.

[7] W. W. Peterson and D. T. Brown, "Cyclic codes for error detection," *Proceedings of the IRE*, vol. 49, no. 1, pp. 228–235, 1961.

[8] R. Gallager, "Low-density parity-check codes," *IRE Transactions on information theory*, vol. 8, no. 1, pp. 21–28, 1962.

[9] Z. Alkhalifa, V. S. Nair, N. Krishnamurthy, and J. A. Abraham, "Design and evaluation of system-level checks for on-line control flow error detection," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 6, pp. 627–641, 1999.

[10] A. Mahmood and E. J. McCluskey, "Concurrent error detection using watchdog processors-a survey," *IEEE Transactions on Computers*, vol. 37, no. 2, pp. 160–174, 1988.

[11] L. A. Clarke and D. S. Rosenblum, "A historical perspective on runtime assertion checking in software development," *ACM SIGSOFT Software Engineering Notes*, vol. 31, no. 3, pp. 25–37, 2006.

[12] N. Delgado, A. Q. Gates, and S. Roach, "A taxonomy and catalog of runtime software-fault monitoring tools," *IEEE Transactions on software Engineering*, vol. 30, no. 12, pp. 859–872, 2004.

[13] X. Iturbe, B. Venu, E. ?zer, J.-L. Poupat, G. Gimenez, and H.-U. Zurek, "The Arm Triple Core Lock-Step (TCLS) Processor," *ACM Transactions on Computer Systems*, vol. 36, pp. 1–30, 06 2019.

[14] E. Rotenberg, "AR-SMT: a microarchitectural approach to fault tolerance in microprocessors," in *Digest of Papers. Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing (Cat. No.99CB36352)*, 1999, pp. 84–91.

[15] E. Talpes, D. D. Sarma, G. Venkataramanan, P. Bannon, B. McGee, B. Floering, A. Jalote, C. Hsiong, S. Arora, A. Gorti, and G. S. Sachdev, "Compute Solution for Tesla's Full Self-Driving Computer," *IEEE Micro*, vol. 40, no. 2, pp. 25–35, 2020.

[16] S. K. Reinhardt and S. S. Mukherjee, "Transient fault detection via simultaneous multithreading," in *Proceedings of 27th International Symposium on Computer Architecture (IEEE Cat. No.RS00201)*, 2000, pp. 25–36.

[17] S. S. Mukherjee, M. Kontz, and S. K. Reinhardt, "Detailed design and evaluation of redundant multi-threading alternatives," in *Proceedings 29th Annual International Symposium on Computer Architecture*, 2002, pp. 99–110.

[18] M. Gomaa, C. Scarbrough, T. N. Vijaykumar, and I. Pomeranz, "Transient-fault recovery for chip multiprocessors," in *30th Annual International Symposium on Computer Architecture, 2003. Proceedings.*, 2003, pp. 98–109.

[19] C. LaFrieda, E. Ipek, J. F. Martinez, and R. Manohar, "Utilizing Dynamically Coupled Cores to Form a Resilient Chip Multiprocessor," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, 2007, pp. 317–326.

[20] H. Jeon and M. Annavaram, "Warped-DMR: Light-weight Error Detection for GPGPU," in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012, pp. 37–47.

[21] J. P. J. Kelly, T. I. McVittie, and W. I. Yamamoto, "Implementing design diversity to achieve fault tolerance," *IEEE Software*, vol. 8, no. 4, pp. 61–71, 1991.

[22] A. Avizienis, "The N-version approach to fault-tolerant software," *IEEE Transactions on software engineering*, no. 12, pp. 1491–1501, 1985.

[23] P. E. Ammann and J. C. Knight, "Data diversity: An approach to software fault tolerance," *IEEE Transactions on Computers*, vol. 37, no. 4, pp. 418–425, 1988.

[24] H. Zhao, S. K. S. Hari, T. Tsai, M. B. Sullivan, S. W. Keckler, and J. Zhao, "Suraksha: A framework to analyze the safety implications of perception design choices in avs," in *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE Computer Society, 2021.

[25] NVIDIA, "NVIDIA DRIVE | NVIDIA Developer," https://developer.nvidia.com/driveworks.

[26] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, "Learning by Cheating," in *Conference on Robot Learning (CoRL)*, 2019.

[27] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An Open Urban Driving Simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[28] T. Tsai, S. K. S. Hari, M. B. Sullivan, O. Villa, and S. W. Keckler, "NVBitFI: Dynamic Fault Injection for GPUs," in *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks*, 2021.

[29] "PinFI," https://github.com/DependableSystemsLab/pinfi.

[30] F. F. d. Santos, S. K. S. Hari, P. M. Basso, L. Carro, and P. Rech, "Demystifying gpu reliability: Comparing and combining beam experiments, fault simulation, and profiling," in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2021, pp. 289–298.

[31] S. K. S. Hari, T. Tsai, M. Stephenson, S. W. Keckler, and J. Emer, "SASSIFI: An architecture-level fault injection tool for gpu application resilience evaluation," in *Performance Analysis of Systems and Software (ISPASS), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 249–258.

[32] B. Fang, K. Pattabiraman, M. Ripeanu, and S. Gurumurthi, "GPU-Qin: A methodology for evaluating the error resilience of gpgpu applications," in *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*. IEEE, 2014, pp. 221–230.

[33] Q. Lu, M. Farahani, J. Wei, A. Thomas, and K. Pattabiraman, "Llfi: An intermediate code-level fault injection tool for hardware faults," in *2015 IEEE International Conference on Software Quality, Reliability and Security*, 2015, pp. 11–16.

[34] "NVBitFI," https://github.com/NVlabs/nvbitfi.

[35] S. Mitra, N. Saxena, and E. McCluskey, "A design diversity metric and reliability analysis for redundant systems," in *International Test Conference 1999. Proceedings (IEEE Cat. No.99CH37034)*, 1999, pp. 662–671.

[36] Baidu, "Apollo 5.0," https://github.com/ApolloAuto/apollo.

[37] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets Robotics: The KITTI Dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[38] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The Kitti Vision Benchmark Suite," in *, 2012 IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 3354–3361.

[39] National Highway Traffic Safety Administration (NHTSA), "Pre-Crash Scenario Typology for Crash Avoidance Research DOT HS 810 767," 2007.

[40] "Carla autonomous driving challenge." [Online]. Available: https://leaderboard.carla.org/challenge/

[41] J. Wei, A. Thomas, G. Li, and K. Pattabiraman, "Quantifying the Accuracy of High-Level Fault Injection Techniques for Hardware Faults," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2014, pp. 375–382.

[42] Jessie White Secretary of State, "Illinois rules of the road," https://www.ilsos.gov/publications/pdf_publications/dsd_a112.pdf, 2021.

[43] D. B. Fambro, *Determination of stopping sight distances (Report / National Cooperative Highway Research Program)*. National Academy Press, 1997.

[44] R. Moore-Colyer, "Nvidia Xavier Supercomputer Aims To Turn Cars Into AIs On Wheels."

[45] Z. Chen, G. Li, and K. Pattabiraman, "A low-cost fault corrector for deep neural networks through range restriction," 2021.

[46] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 2828–2837. [Online]. Available: https://doi.org/10.1145/3292500.3330672

[47] G. L. Hicks, L. D. Howe Jr, and F. A. Zurla Jr, "Instruction retry mechanism for a data processing system," Aug. 23 1977, uS Patent 4,044,337.

[48] A. Mahmoud, S. K. S. Hari, M. B. Sullivan, T. Tsai, and S. W. Keckler, "Optimizing software-directed instruction replication for GPU error detection," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2018, pp. 842–853.

[49] K. S. Yim, C. Pham, M. Saleheen, Z. Kalbarczyk, and R. Iyer, "Hauberk: Lightweight silent data corruption error detector for gpgpu," in *2011 IEEE International Parallel & Distributed Processing Symposium*. IEEE, 2011, pp. 287–300.

[50] T. J. Dell, "A white paper on the benefits of chipkill-correct ECC for PC server main memory," *IBM Microelectronics division*, vol. 11, pp. 1–23, 1997.

[51] J. E. Barth Jr, C. E. Drake, J. A. Fifield, W. P. Hovis, H. L. Kalter, S. C. Lewis, D. J. Nickel, C. H. Stapper, and J. A. Yankosky, "Dynamic ram with on-chip ecc and optimized bit and word redundancy," Jul. 28 1992, uS Patent 5,134,616.

[52] M. B. Sullivan, S. K. S. Hari, B. Zimmer, T. Tsai, and S. W. Keckler, "SwapCodes: Error Codes for Hardware-Software Cooperative GPU Pipeline Error Detection," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 762–774.

[53] R. Nathan and D. J. Sorin, "Argus-G: Comprehensive, Low-Cost Error Detection for GPGPU Cores," *IEEE Computer Architecture Letters*, vol. 14, no. 1, pp. 13–16, 2015.

[54] B. H. Meyer, B. H. Calhoun, J. Lach, and K. Skadron, "Cost-effective safety and fault localization using distributed temporal redundancy," in *2011 Proceedings of the 14th International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*, 2011, pp. 125–134.

[55] J. Fu, Q. Yang, R. Poss, C. R. Jesshope, and C. Zhang, "On-demand thread-level fault detection in a concurrent programming environment," in *2013 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, 2013, pp. 255–262.

[56] D. Tiwari, S. Gupta, J. Rogers, D. Maxwell, P. Rech, S. Vazhkudai, D. Oliveira, D. Londo, N. DeBardeleben, P. Navaux *et al.*, "Understanding GPU errors on large-scale HPC systems and the implications for system design and operation," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2015, pp. 331–342.

[57] C. Di Martino, Z. Kalbarczyk, R. K. Iyer, F. Baccanico, J. Fullop, and W. Kramer, "Lessons learned from the analysis of system failures at petascale: The case of Blue Waters," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2014, pp. 610–621.

[58] K.-H. Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE transactions on computers*, vol. 100, no. 6, pp. 518–528, 1984.

[59] S. K. S. Hari, M. B. Sullivan, T. Tsai, and S. W. Keckler, "Making Convolutions Resilient via Algorithm-Based Error Detection Techniques," *arXiv preprint arXiv:2006.04984*, 2020.

[60] D. Gizopoulos, M. Psarakis, S. V. Adve, P. Ramachandran, S. K. S. Hari, D. Sorin, A. Meixner, A. Biswas, and X. Vera, "Architectures for online error detection and recovery in multicore processors," in *2011 Design, Automation & Test in Europe*. IEEE, 2011, pp. 1–6.

[61] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini, "Anomaly detection using autoencoders in high performance computing systems," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 9428–9433.

[62] F. Haas, S. Weis, T. Ungerer, G. Pokam, and Y. Wu, "Fault-Tolerant Execution on COTS Multi-core Processors with Hardware Transactional Memory Support," in *ARCS*, 03 2017, pp. 16–30.

[63] M. S. Alhakeem, P. Munk, R. Lisicki, H. Parzyjegla, H. Parzyjegla, and G. Muehl, "A Framework for Adaptive Software-Based Reliability in COTS Many-Core Processors," in *ARCS 2015 - The 28th International Conference on Architecture of Computing Systems. Proceedings*, 2015, pp. 1–4.

[64] D. J. Scales, M. Nelson, and G. Venkitachalam, "The Design of a Practical System for Fault-Tolerant Virtual Machines," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 4, p. 30?39, Dec. 2010. [Online]. Available: https://doi.org/10.1145/1899928.1899932

[65] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, and D. I. August, "Swift: software implemented fault tolerance," in *International Symposium on Code Generation and Optimization*, 2005, pp. 243–254.

[66] K. Pattabiraman, Z. Kalbarczyk, and R. K. Iyer, "Application-based metrics for strategic placement of detectors," in *11th Pacific Rim International Symposium on Dependable Computing (PRDC'05)*. IEEE, 2005, pp. 8–pp.

[67] K. Pattabiraman, N. Nakka, Z. Kalbarczyk, and R. Iyer, "SymPLFIED: Symbolic program-level fault injection and error detection framework," in *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*. IEEE, 2008, pp. 472–481.

[68] S. Alcaide, L. Kosmidis, C. Hernandez, and J. Abella, "Software-only based Diverse Redundancy for ASIL-D Automotive Applications on Embedded HPC Platforms," in *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2020, pp. 1–4.

[69] A. Shye, J. Blomstedt, T. Moseley, V. J. Reddi, and D. A. Connors, "PLR: A Software Approach to Transient Fault Tolerance for Multicore Architectures," *IEEE Transactions on Dependable and Secure Computing*, vol. 6, no. 2, pp. 135–148, 2009.

[70] G. Li, K. Pattabiraman, C.-Y. Cher, and P. Bose, "Understanding error propagation in GPGPU applications," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2016, pp. 240–251.