Strategic Resilience Evaluation of Neural Networks within Autonomous Vehicle Software

Anna Schmedding[1], Philip Schowitz[2], Xugui Zhou[3], Yiyang Lu[1], Lishan Yang[4], Homa Alemzadeh[3], Evgenia Smirni[1]

[1] William & Mary, USA, {akschmedding,ylu21,exsmir}@wm.edu
[2] University of British Columbia, Canada, philipns@cs.ubc.ca
[3] University of Virginia, USA, {xz6cz,ha4d}@virginia.edu
[4] George Mason University, USA, lyang28@gmu.edu

Abstract. Self-driving technology has become increasingly advanced over the past decade due to the rapid development of deep neural networks (DNNs). In this paper, we evaluate the effects of transient faults in DNNs and present a methodology to efficiently locate critical fault sites in DNNs deployed within two cases of autonomous vehicle (AV) agents: Learning by Cheating (LBC) and OpenPilot. We locate the DNN fault sites using a modified Taylor criterion and strategically inject faults that can affect the functioning of AVs in different road and weather scenarios. Our fault injection methodology identifies corner cases of DNN vulnerabilities that can cause hazards and accidents and therefore dramatically affect AV safety. Additionally, we evaluate mitigation mechanisms of such vulnerabilities for both AV agents and discuss the insights of this study.

Keywords: Autonomous Vehicles, Fault Tolerance, DNNs

1 Introduction

Autonomous vehicles (AVs) are real-world safety-critical systems of increasing importance. With the growing complexity of software and use of deep neural networks (DNNs) for perception and control in AVs, many factors can threaten their safe operation, such as software bugs [1] and transient faults in hardware [2], leading to mis-classifications by DNNs and potential safety hazards.

Transient faults (i.e., soft errors) originating from cosmic radiation [3] or from operating under low voltage [4] have been shown to threaten the functionality of DNN hardware and software [5]. Transient faults in the main memory (DRAM) can manifest as single- or multi-bit flips, specifically in neurons or weights of DNN models [5] and may cause silent data corruption (SDC) where the output is faulty despite a seemingly "correct" execution. Since DRAM is used in AVs[5], this safety-critical application inherits the reliability challenges of DRAM faults. Transient faults have already contributed to vehicle crashes [7].

[5] For example, the NVIDIA Jetson AGX Orin Series, which is used by NVIDIA DRIVE, uses 32GB or 64GB of DRAM [6]

	LBC	Supercombo
# **CONV Layers**	40	70
# **Weights**	21,268,928	5,811,616
# **Single-Bit Fault Sites**	680,605,696	185,971,712
# **Double-Bit Fault Sites**	21,098,776,576	5,765,123,072
# **Triple-Bit Fault Sites**	632,963,297,280	172,953,692,160

Table 1: Fault Space for OpenPilot Supercombo and LBC.

Locating critical faults that cause safety hazards or accidents is necessary in such safety-critical systems. A major challenge here is the vast fault site space, often in the order of billions (see Table 1) which would require thousands of years to exhaustively analyze. This makes identifying corner cases where faults may affect the functional safety of a self-driving vehicle [8] similar to searching for a needle in a haystack. Within the AV ecosystem, the classical statistical fault injection [9], cannot discover the critical corner cases that could lead to safety violations. Past works focus on specific DNN tasks (e.g., image classification) and examine DNN resilience *without the context of the entire AV system* [10, 11]. Recent AV resilience assessment works have focused on input, models, or outputs [12–14], but not on its DNN components that are at the core of AV operation. *Our aim is to develop a method to efficiently locate these critical fault sites in the DNN components of AVs.* We evaluate these fault sites by injecting transient faults in DNN weights and determining whether their effects propagate to other AV components and eventually result in hazards or accidents.

We present a *strategic fault injection* method, called *Taylor-Guided Fault Injection (TGFI)*, that identifies and targets the DNN weights that are of high importance to reliable inference. This is done using a modified Taylor criterion [15] which ranks all the DNN weights with respect to their relative importance to inference accuracy. We inject faults in those important weights and show that our strategic fault injection method can efficiently discover safety-critical vulnerabilities in AVs. We specifically focus on locating critical fault sites within two AV systems: (i) Learning-by-Cheating (LBC), a fully autonomous self-driving agent [16] that is widely used in academic studies [2], and (ii) OpenPilot, a popular driver-assistance system that is used in over 250 existing car models on the road [17]. By using two systems with different levels of autonomy, we demonstrate the ability of TGFI to generalize to different AV DNNs.

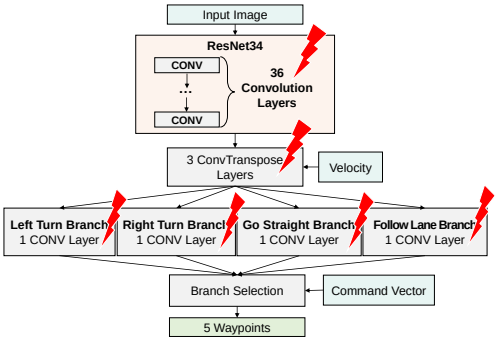
We also characterize the effect of mitigation in the cases where these critical faults occur. For LBC, we consider a state-of-the-art fault tolerance method based on neuron value range restriction for CNNs, called Ranger [11]. For OpenPilot, we examine the existing system safety checks which return control to the human driver. Additionally, we examine the effects of considering contextual factors such as the location of faults and environmental conditions that impact the input in order to offer insights into the practical reliability challenges of deploying AVs on the road. Both mitigation methods show improvement in resilience, while TGFI is still able to find critical corner cases.

2 Autonomous Driving Frameworks

The Society of Automotive Engineers (SAE) defines 6 levels of driving automation for AVs, from Level 0 (L0, no driving automation) to Level 5 (L5, full driving automation) [18]. L0 to L2 assume that there is a human in the loop who controls the automotive environment by supervising or taking over the autonomous features. For higher levels, the car autonomously controls the driving environment without human involvement. With DNN models incorporated, an L4 AV may use end-to-end ML models for perception and planning without human intervention, while L0-L2 levels use DNNs for driver assistance.

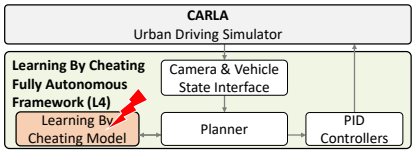
2.1 L4 System: LBC

Learning by Cheating (LBC) is a pretrained end-to-end agent for L4 autonomous driving and is widely used in research studies [2,19,20]. Fig. 1a shows the model structure of LBC, which is built on a ResNet34 [21] backbone to process input images from a front-facing camera on the vehicle. The model takes two additional inputs: vehicle speed and a high-level command vector generated by the planner, instructing the vehicle to follow the lane, turn left/right or go straight at an intersection. After the ResNet34 backbone, the model splits into four parallel branches that each corresponds to a high-level command. The final output is a set of five waypoints of the AV path (see Fig. 1b), which are passed to a low-level controller that produces the steering, throttle, and braking commands.



(a) LBC model structure.

(b) LBC screenshot. Waypoints are red dots.

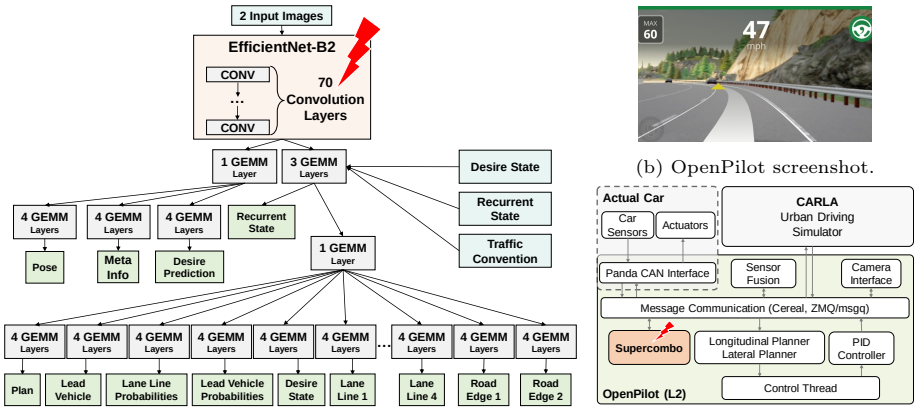


(c) LBC integration in the Carla simulator.

Fig. 1: LBC framework.

2.2 L2 System: OpenPilot

OpenPilot is an L2 Autonomous Driving Assistance System (ADAS) that supports more than 250 popular makes and models of cars [17]. A high-level overview of OpenPilot is shown in Fig. 2c. Car sensor data such as images and vehicle state information are passed into the Supercombo model. The output of Supercombo is sent to the planners. The PID Controller uses the results from the planner to decide actuator actions. The generated commands (e.g., brake) are passed through the Panda CAN interface to the actuators to perform the commands.



(a) OpenPilot Supercombo model structure.

(b) OpenPilot screenshot.

(c) OpenPilot Overview.

Fig. 2: OpenPilot framework.

The Supercombo model is at the core of the perception module and provides fifteen output fields. As shown in Fig. 2a, Supercombo utilizes an EfficientNet-B2 [22] base CNN for processing the incoming images from the car sensors. Then it uses additional inputs of the traffic convention, desire state, and recurrent state to incorporate the state of the vehicle and environment. Once all inputs have been tied in, Supercombo branches into separate general matrix multiplication (GEMM) computations to generate the plan, lanelines, laneline probabilities, road edges, lead vehicles, lead probabilities, desire state, meta information, vehicle pose, and recurrent state. Fig. 2b shows an example screenshot.

2.3 Driving Simulator: CARLA

CARLA is an open-source simulator for autonomous driving research, design, and testing [23]. It provides a realistic urban environment for a vehicle to navigate with features such as variable road and weather conditions. CARLA provides a wealth of data at each simulation timestamp including whether a collision, lane invasion, or red light violation has occurred. CARLA is integrated to Openpilot and LBC, see Fig. 2c and Fig. 1c, respectively.

3 Methodology

Fault Model. We use fault injection in a single 32-bit floating point weight of the DNN to simulate commonly occurring transient faults in DRAM (Dynamic Random Access Memory). DNN weights typically reside in DRAM. A fault site in a neural network weight is defined by the weight id and the bit position(s) of the weight to be flipped. The size of the fault site space for single-, double-, and triple-bit faults for OpenPilot Supercombo and LBC is tremendous, see Table 1 and prevents its exhaustive exploration.

For the majority of the analysis (Sections 4 and 5) we focus on double-bit flips that are detectable but not correctable by ECC. This is consistent with other reliability studies [11, 24] and also consistent with DRAM faults in the

wild [25]. For a broader view of the effect of bit flips, we also do experiments with single-bit flip (detectable and correctable) and triple-bit flips (undetectable, their safety implications are similar to double-bit ones, see Section 6).

Fault Injection Method. Fault injection is implemented as a two-stage process: *Preparation* and *Injection*. In the *Preparation* stage, before the actual simulation run, we load the (correct) neural network and select an injection site. Portions of the network where faults may be injected are denoted by a lightning bolt in Fig. 1a for LBC and Fig. 2a for OpenPilot Supercombo. We corrupt a weight tensor of the neural network by altering the values of an individual weight (depending on the type of experiment, we induce one single-bit, one double-bit, or one triple-bit fault), and save the corrupted tensor. [6] In the *Injection* stage, the corrupted model generated by the fault injector is used by the AV control software while performing the driving task. The corrupted Pytorch or ONNX files are loaded at the beginning of each LBC or OpenPilot experiment.

Fault Injection Outcomes. Throughout the simulation, we focus on events that indicate abnormal behavior. A *lane invasion* occurs when the vehicle crosses into a neighboring lane erroneously. Since a lane invasion can occur when the car barely crosses the lane line, a lane invasion alone is not a hazard. We define a *hazard* to be one of the following situations:

H1: The vehicle violates safe following-distance constraints with the lead vehicle: *Relative Distance / Speed* $\leqslant t_{safe}$ and *Speed* $>$ *Lead Speed*.

H2: The vehicle drives out of lane beyond a threshold (e.g., 0.1 meter) while speed is higher than β, these are predetermined values given by the driving scenario.

We record all hazards that occur within the experiment as well as their time stamps. These hazards can lead to the following *accidents*:

A1: Collision with the lead vehicle.

A2: Collision with road-side objects or other vehicles in the neighboring lane.

An accident terminates the simulation, at which point the time stamp and the nature of the accident are recorded. The simulation also terminates if the vehicle successfully reaches its goal location. These definitions are based on the STPA [28] hazard analysis method, which has been utilized in other studies [12, 29]. The hazards considered here are indicative of failures of lane keep assist (LKA) and adaptive cruise control (ACC), two main functionalities of L2 AVs.

3.1 Vulnerable Weights: Taylor Guided Fault Injection (TGFI)

In a fault space as vast as the one reported on Table 1, the odds that the standard practice of 1,000 random fault injections [30] capture rare corner cases that result

[6] Once the weight is altered, the modified model is written to a file in the necessary format (ONNX [26] or PyTorch [27], for OpenPilot and LBC, respectively). To corrupt Pytorch models, we load the model, alter the state dictionary associated with it and then save the new faulty model. For ONNX models, we read the model file as bytes, locate the plain text layer ID, and modify the bits corresponding to the target weight in the binary data.

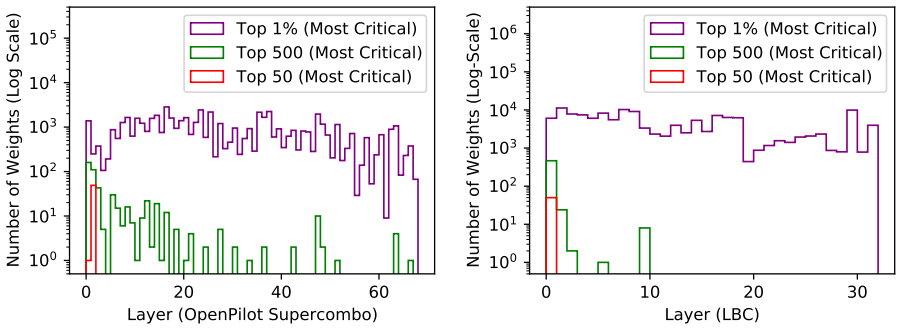


Fig. 3: Locations of top 50, top 500, and top 1% of important weights in OpenPilot Supercombo (left) and LBC (right).

in catastrophic driving scenarios are low. Since our target is the identification of the aforementioned corner cases, we rank the importance of the weights in the neural network using a modified Taylor criterion [15] which estimates the first-order Taylor expansion of the contribution of each weight to the accuracy of the neural network. The more a weight contributes to the accuracy of the network, the more critical it is and the more it can affect accuracy if a fault occurs there. We relatively rank all weights in the two target models using the *importance scores* generated by the Taylor criterion [15]. The importance of a weight using this criterion can be estimated using the following equation:

$$I_m^{(1)}(W) = (g_m w_m)^2, \tag{1}$$

where I is the importance, W is the set of network parameters, g_m are elements of the gradient, and w_m are the weight values.

We use the Comma2k19 data set [31] of real-world driving footage as input to OpenPilot Supercombo and LBC, and compute the importance of their weights [15]. Figure 3 illustrates the location (layer, x-axis) and number (y-axis logscale) of the most critical weights for OpenPilot Supercombo and LBC. We make the following observations: 1) critical weights may be located in any layer and 2) the few *most* critical weights are concentrated in the earlier layers of both models. We perform fault injection experiments on the most critical weights as guided by Equation 1, called *Taylor-Guided Fault Injection (TGFI)*.

3.2 Experimental Campaigns

Every distinct fault site uses the same map for the AV to traverse, the same starting location of the car(s) in the simulator, the same goal location, and the same random seed, initial velocity, and weather. The fault sites are selected according to the fault injection (FI) campaign: 1) **Random**: The weight where the double bit flip occurs is chosen using a uniform distribution [30]: we select 1,000 random fault sites to obtain results with 95% confidence intervals and ±3% error margins. 2) **TGFI-top500**: We select the top 500 most critical weights. 3) **TGFI-top50**: We select the top 50 most critical weights.

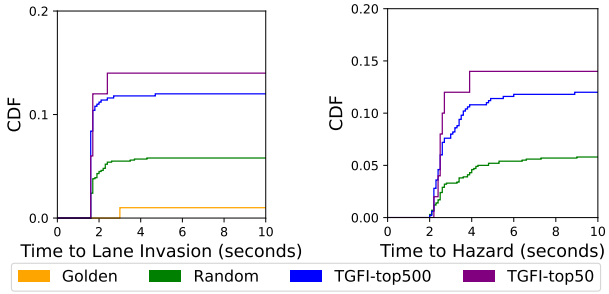


Fig. 4: LBC time to lane invasion and time to hazard. No time to driver intervention because of L4 autonomy. *Note that CDFs do not reach 1.0 since a portion of the experiments do not experience a lane invasion or hazard.*

4 Resilience Evaluation

Following the methodology laid out in Section 3, we perform three FI campaigns for LBC and OpenPilot. These FI campaigns highlight different methods for selecting the target weight for fault injection (i.e., Random FI, TGFI-top500, TGFI-top50) using a two-bit fault model. We perform golden runs (i.e., fault-free) to capture normal behavior. The driving scenario is on a curved road in cloudy weather, which represents ideal driving conditions (i.e., good visibility with no glare). Variations of experimental setup are explored in Section 6.

4.1 Resilience of L4 LBC

Lane invasions are the least severe violation, since not every lane invasion leads to a hazard. Only 1% of experiments have lane invasions in golden runs, indicating that these are rare in fault-free cases. The time to lane invasion is shown in Fig. 4(a). Across all three fault injection campaigns, the lane invasion tends to occur early in the simulation. Table 2 shows the percentage: TGFI experiments double the percentage of lane invasions comparing to the random FI experiments.

ADS	Fault-Site Selection	H1	H2	A1	A2	Lane Inv.	Driver Int.
L4 (LBC)	Golden run	N/A	0%	N/A	0%	1%	N/A
	Random FI	N/A	5.9%	N/A	5.9%	6%	N/A
	TGFI-top500	N/A	12%	N/A	12%	12%	N/A
	TGFI-top50	N/A	14%	N/A	14%	14%	N/A
L2 (OpenPilot Supercombo)	Golden run	0%	0%	0%	0%	10%	0%
	Random FI	2.0%	0.6%	0%	0.1%	23.4%	21.4%
	TGFI-top500	3.3%	8.8%	0.1%	0.2%	29.62%	21.0%
	TGFI-top50	7.9%	16.3%	0.2%	0.3%	62.5%	18.3%

Table 2: Percentage of various events. There is no front vehicle in LBC, hence H1 and A1 cannot happen. Driver intervention: the L2 system returns control to the driver.

No hazards are detected in the golden fault-free runs, but Random FI and TGFI cause H2 hazards. The time to hazard is shown in Fig. 4. For both random FI and TGFI, hazards are encountered between the one and two second

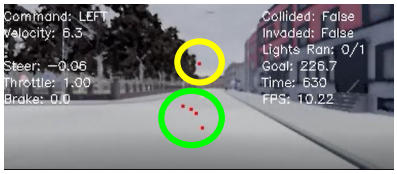


Fig. 5: Fault injection in left-turn control command branch in LBC. Four waypoints (circled in green) are correct but one (circled in yellow) is incorrect.

mark. There is a near-complete overlap between the experiments that have lane invasions and the experiments that have hazards, i.e., if the fault causes a lane invasion, then the fault is also severe enough to cause a hazard shortly after. Similar to the lane invasions, TGFI finds more hazards than Random FI.

Every hazard for Random FI and TGFI scenarios results in an accident, indicating that LBC has a very limited ability to function under critical faults. Indeed, LBC does not have alerts or safety checks. TGFI triggers more accidents than Random FI.

We also evaluate the impact of faults in the convolution branches at the end of the network (see Fig. 1a) corresponding to high-level control commands. Fig. 5 presents the results of a fault injected in the branch corresponding to the left turn control command: four waypoints are correct, but one (yellow circle) is clearly incorrect. When the car turns the corner, it tries to adhere to the trajectory generated by fitting a curve to the points, but in doing so turns the corner too widely and crashes into the wall on the side of the road.

4.2 Resilience of L2 OpenPilot

In OpenPilot, lane invasions occur often but do not always result in a hazard or accident. 10% of golden runs and 23.4% of Random FI experiments have lane invasions, see Table 2. TGFI-top500 shows slightly more lane invasions and TGFI-top50 nearly triples the number of lane invasions compared to Random FI. The time to lane invasion is shown in Fig. 6(a): most of the lane invasions occur between the 15 and 20 second marks.

The golden runs never result in a hazard, but 2.6% of Random FI experiments result in a hazard that typically occurs between the 5 and 10 second marks,

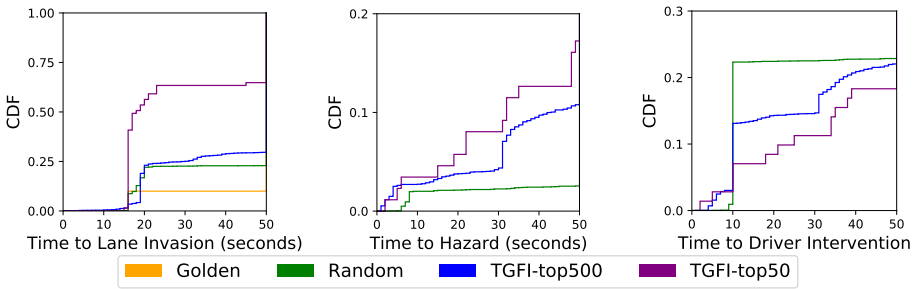


Fig. 6: OpenPilot Supercombo: Time to lane invasion, hazard, and driver intervention. No hazards or driver interventions were observed in golden runs. *CDFs do not reach 1.0 since some experiments do not have a lane invasion, hazard, or driver intervention.*

see Fig. 6(b). 10.8% of TGFI-top500 experiments and 22.7% of TGFI-top50 experiments result in a hazard (1.3% and 1.5% of them experience both H1 & H2 hazards for TGFI-top500 and TGFI-top50, respectively). Hazards only occur after second 15 in the simulation and show a steeper increase towards second 50. The H1 hazard that occurs when the vehicle follows another car too closely occurs in 7.9% of simulations for TGFI-top50 and the H2 hazard, which indicates a significant lane invasion, occurs in 16.3% of experiments for TGFI-top50, see Table 2. TGFI results in more hazards than than Random FI.

Since OpenPilot requires the driver to resume control of the vehicle in the case of dangerous situations, hazards are often masked and accidents rarely happen, see Table 2. We will discuss this mitigation technique in detail in 5.2.

5 Mitigation

The effects of faults can be mitigated through several approaches, many of which are orthogonal to one another [11, 24, 32]. In this section, we examine different mitigation techniques for the two AV cases examined here.

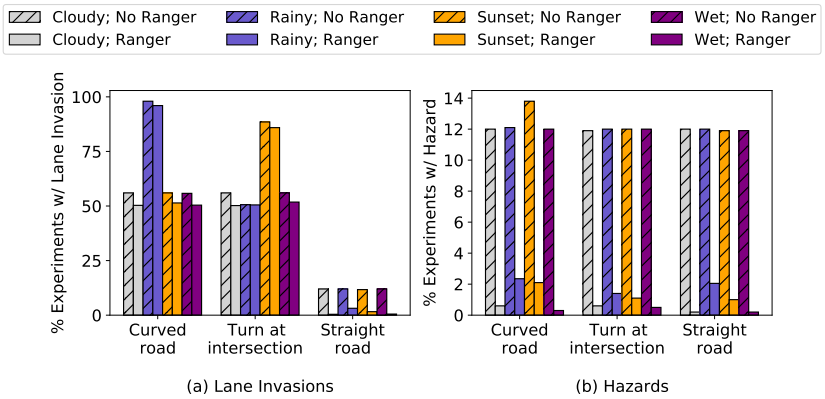
5.1 L4 LBC: Ranger

Ranger [11] is a popular, state-of-the-art fault corrector which employs range restriction on neuron activation values to protect ML models from faults with negligible overhead. Here, we present a proof-of-concept mitigation of applying Ranger to LBC. To implement Ranger, we insert range restriction into the model following activation layers at crucial points in the network, such as after convolution layers. Each protection layer has a pair of minimum and maximum activation values to use as bounds, which are set after profiling through golden runs under cloudy (ideal) weather. This step is performed once, before the deployment of the protected model with Ranger. When Ranger is active, any activation values outside the ranges defined by Ranger are clipped to the bound.

We examine the effectiveness of Ranger using the TGFI-top500 experiments, see Fig.7. We examine three CARLA driving scenarios: curved road, turn at intersection, and straight road. A combination of clear and inclement weather conditions (cloudy, rainy, sunset, wet) is also used to offer insight into how well the fault mitigation functions in unseen conditions. Ranger improves the resiliency of LBC across all three driving scenarios and all weather conditions. Significant improvements are observed under cloudy (ideal) weather conditions.

5.2 L2 OpenPilot: Driver Intervention

As an L2 ADAS, OpenPilot raises driver alerts and returns control to the driver when a problem is detected by its safety checker, i.e., *driver intervention*. We record the *time to driver intervention* as a CDF in Fig. 6(c). The



(a) Lane Invasions (b) Hazards

Fig. 7: Comparison of LBC without Ranger vs. LBC with Ranger for the TGFI-500 scenario.

golden runs never require driver intervention and are therefore not plotted. Random FI results in driver intervention in 21.4% of experiments, occurring close to the 10 second mark. The TGFI-top500 and TGFI-top50 experiments show an initial jump in driver intervention early in the simulation. OpenPilot has 3 types of driver alerts which may be displayed when control of the vehicle is returned to the driver, see Table 3. In fault injection experiments, PlannerError/noEntry alerts are the most common ones. These alerts indicate that the Model Predictive Control (MPC) cannot find a feasible solution for lateral planning (steering) and longitudinal planning (gas/brake) and releases control to the driver, successfully mitigating the fault before any hazard is triggered. The CanError/ImmediateDisable alerts indicate communication errors and occur the least frequently. The steerSaturatedWarning alerts also rarely occur and they indicate that the car is swerving sharply in the presence of the fault. *This indicates that TGFI finds faults that are more rare and more challenging for OpenPilot safety checks to detect and mitigate.*

Table 3: Experiments where the driver is alerted to a problem in OpenPilot.

Alert	**Golden Runs**	**Random**	**TGFI-top500**	**TGFI-top50**
plannerError/noEntry	0%	19.8%	10.88%	4.01%
canError/immediateDisable	0%	0.5%	0.1%	0.4%
steerSaturated/warning	0%	1.9%	2.98%	3.85%

6 Case studies and Discussion

In this section, we use LBC as a case study to evaluate and discuss the impact of different faults, layers of DNN and bit positions on AV resilience.

6.1 Importance of Layer Depth for Resilience

Table 4 shows four distinct experiments with corrupted LBC models that identify the importance of the layer depth where the fault occurs. When Ranger is not applied and if the fault is injected in the earlier DNN layers (experiments

Exp. ID	Layer ID	Lane Invasion (No Ranger)	Accidents (No Ranger)	Lane Invasion (Ranger)	Accidents (Ranger)
1	10	100%	100%	0%	0%
2	30	100%	100%	0%	0%
3	37	92.5%	100%	91%	100%
4	38	53.3%	84.7%	32.7%	78.4%

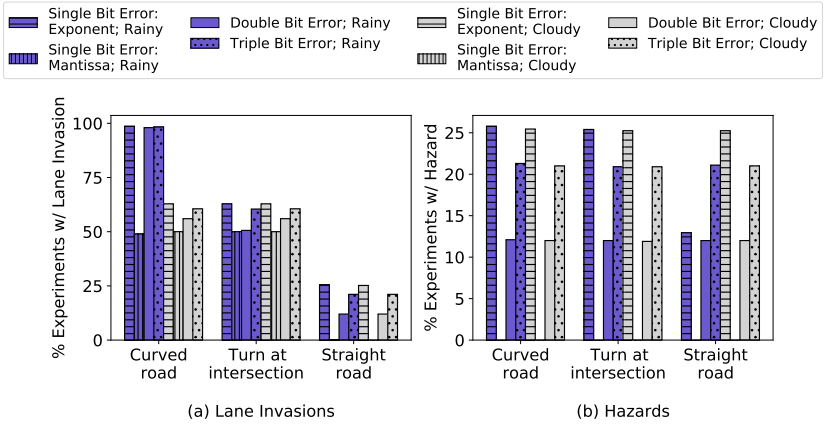
Table 4: Case study of four distinct corrupted LBC models.

1 and 2), resilience is severely affected: all faults result to accidents. Faults occurring earlier in the network have more time to propagate horizontally across the neurons, resulting in more severe corruption. This propagation still occurs even if the fault site is relatively deep into the ResNet34 section of the network, as experiment 2 with injection in layer 30 shows. The fault sites of experiments 3 and 4 are in a the final layers in the network, therefore error propagation is minimal and the severity of corruption in the final output is lessened.

With Ranger, results for faults injected in layer 10 and 30 improve from 100% accidents to 0% accidents. For faults in layer 37 and 38, applying Ranger still results in a majority of accidents. Since these two layers are at the end of the network, the clipped Ranger bound used as output, differs significantly from the ideal one.

6.2 Sensitivity to Single- and Multi-bit Faults

We analyze the sensitivity of LBC to single- and multi-bit faults [33]. We evaluate how the bit position(s) and the number of bit flips affect hazards in the LBC model under cloudy ("easy") and rainy ("challenging") weather conditions using TGFI-500, as shown in Fig. 8. We compare fault injection outcomes using different numbers and locations of bit flips: single-bit exponent, single-bit mantissa, double-bit (the fault model majorly used in this paper), and triple-bit. These experiments show that single-bit exponent causes the most hazards, followed by triple-bit, double-bit, and finally single-bit mantissa. This experiment also shows that lane invasions and hazards for *non-detectable triple* bit faults are prominent, thus their mitigation for safety is of paramount importance.



(a) Lane Invasions (b) Hazards

Fig. 8: Bit position analysis for LBC and the TGFI-top500 experimental campaign.

6.3 Lessons Learned from L4 LBC and L2 OpenPilot

L4 LBC and the L2 OpenPilot have similarities: both ML models are structured around a backbone CNN which is more vulnerable to faults in earlier layers than later ones. Faults are commonly masked when they occur in weights of low importance. Meanwhile, structural differences of the DNNs affect their resilience: LBC has high-level command branches that activate or deactivate parts of the network depending on the driving scenario, thus possibly masking faults during inference. OpenPilot uses all layers in every inference, meaning that faults can only be masked by the network itself, rather than circumstances around its use. Secondly, the format of the DNN output is different. LBC outputs waypoints while OpenPilot's Supercombo outputs information about lanelines, road edges, lead vehicle position, and many other variables. The availability of these different outputs in OpenPilot allows for the implementation of safety checks.

For OpenPilot, we find that many accidents are not mitigated by the system safety checks. This speaks to both the need for the improvement of existing checks, and for the implementation of new ones. Adding DNN-oriented error detection/correction mechanisms would improve AV resilience. High-level checks, like many of those employed in OpenPilot, cannot detect potential incorrect outputs but while imperfect, they provide a blueprint for developing resilience in L4 systems like LBC.

7 Related Work

Past studies have shown that faults [34] can cause hazardous behavior in AVs. DeepTest [1] focuses on testing the reliability of autonomous driving systems and safety engineering techniques for autonomous systems are investigated in [35], but these works do not consider soft errors. Other works examine the fault space for AVs through Bayesian fault injection [34] and rely on large amounts of random fault injection experiments. [36] explores the problem of effective safety checks for an ML-based AV.

[2] uses duplication of computation and temporal data diversity to improve the resilience of AVs with LBC but requires additional hardware. On the OpenPilot side, [12] focuses on hazard coverage and fault injection, but does not study its ML model. [37,38] present attacks on the CAN bus or camera input for a vehicle using OpenPilot. Unlike these works, we focus on the effects of unintended faults rather than attacks.

Most importantly, both for LBC and Openpilot, we identify corner cases that are otherwise hard to find: we identify which portions of their DNNs are vulnerable to faults and result in AV safety violations, i.e., we do not simply examine the accuracy of DNNs for classification but their holistic effect into the AV operation.

8 Conclusions

We perform strategic resilience evaluation of the neural networks of two autonomous vehicles against transient faults. We focus on an L4 system widely used in academia and an L2 ADAS system which is widely deployed on the road. We find that both systems are vulnerable to single- and multi-bit faults which may induce hazards/accidents. We use the Taylor criterion to strategically identify the most important weights for reliability in the DNNs used in the L4 LBC and L2 Openpilot and inject errors on those weights using TGFI. TGFI is efficient in identifying vulnerabilities that result in hazards and accidents. We also examine the effectiveness of mitigation in AVs. For L4 self-driving, mitigation techniques such as Ranger can be effective at minimizing the impact of faults. Driver intervention is a crucial contributing factor to the security of L2 systems.

Acknowledgments

This material is based upon work supported by two Commonwealth Cyber Initiative (CCI) grants (#HC-3Q24-047 and COVA C-Q122-WM-02).

References

1. Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th International Conference on Software Engineering*, pages 303–314, 2018.
2. Saurabh Jha, Shengkun Cui, Timothy Tsai, Siva Kumar Sastry Hari, Michael B Sullivan, Zbigniew T Kalbarczyk, Stephen W Keckler, and Ravishankar K Iyer. Exploiting temporal data diversity for detecting safety-critical faults in av compute systems. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 88–100. IEEE, 2022.
3. Vinicius Fratin, Daniel Oliveira, Caio Lunardi, Fernando Santos, Gennaro Rodrigues, and Paolo Rech. Code-dependent and architecture-dependent reliability behaviors. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 13–26. IEEE, 2018.
4. Shrikanth Ganapathy, John Kalamatianos, Bradford M Beckmann, Steven Raasch, and Lukasz G Szafaryn. Killi: Runtime fault classification to deploy low voltage caches without mbist. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 304–316. IEEE, 2019.
5. Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel Emer, and Stephen W Keckler. Understanding error propagation in deep learning neural network (dnn) accelerators and applications. In *Proceedings of Supercomputing*, pages 1–12, 2017.
6. L. S. Karumbunathan. NVIDIA Jetson AGX Orin Series: A Giant Leap Forward for Robotics and Edge AI Applications, 2022. https://www.nvidia.com/content/dam/en-zz/Solutions/gtcf21/jetson-orin/nvidia-jetson-agx-orin-technical-brief.pdf.
7. J. Yoshida. Toyota Case: Single Bit Flip That Killed, 2013. https://www.eetimes.com/toyota-case-single-bit-flip-that-killed.

8. Road vehicles — Functional safety. Standard, International Organization for Standardization, Geneva, CH, December 2018.
9. Régis Leveugle, A Calvez, Paolo Maistri, and Pierre Vanhauwaert. Statistical fault injection: Quantified error and confidence. In *2009 Design, Automation & Test in Europe Conference & Exhibition*, pages 502–506. IEEE, 2009.
10. Fernando Fernandes dos Santos, Pedro Foletto Pimenta, Caio Lunardi, Lucas Draghetti, Luigi Carro, David Kaeli, and Paolo Rech. Analyzing and increasing the reliability of convolutional neural networks on gpus. *IEEE Transactions on Reliability*, 68(2):663–677, 2018.
11. Zitao Chen, Guanpeng Li, and Karthik Pattabiraman. A low-cost fault corrector for deep neural networks through range restriction. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–13. IEEE, 2021.
12. Abu Hasnat Mohammad Rubaiyat, Yongming Qin, and Homa Alemzadeh. Experimental resilience assessment of an open-source driving agent. In *2018 IEEE 23rd Pacific rim international symposium on dependable computing (PRDC)*, pages 54–63. IEEE, 2018.
13. Saurabh Jha, Timothy Tsai, Siva Hari, Michael Sullivan, Zbigniew Kalbarczyk, Stephen W Keckler, and Ravishankar K Iyer. Kayotee: A fault injection-based system to assess the safety and reliability of autonomous vehicles to faults and errors. *arXiv preprint arXiv:1907.01024*, 2019.
14. Saurabh Jha, Shengkun Cui, Subho Banerjee, James Cyriac, Timothy Tsai, Zbigniew Kalbarczyk, and Ravishankar K Iyer. Ml-driven malware that targets av safety. In *2020 50th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pages 113–124. IEEE, 2020.
15. Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11264–11272, 2019.
16. Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Conference on Robot Learning*, pages 66–75. PMLR, 2020.
17. Comma.ai. Supported Cars by OpenPilot. https://github.com/commaai/openpilot/blob/master/docs/CARS.md.
18. SAE International. SAE Levels of Driving Automation™ Refined for Clarity and International Audience. https://www.sae.org/blog/sae-j3016-update, 2021.
19. Angelos Filos, Panagiotis Tigkas, Rowan McAllister, Nicholas Rhinehart, Sergey Levine, and Yarin Gal. Can autonomous vehicles identify, recover from, and adapt to distribution shifts? In *International Conference on Machine Learning*, pages 3145–3153, 2020.
20. Marin Toromanoff, Emilie Wirbel, and Fabien Moutarde. End-to-end model-free reinforcement learning for urban driving using implicit affordances. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7153–7162, 2020.
21. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.
22. Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114, 2019.
23. Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on Robot Learning*, pages 1–16, 2017.

24. Gurunath Kadam, Evgenia Smirni, and Adwait Jog. Data-centric reliability management in gpus. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 271–283. IEEE, 2021.
25. Majed Valad Beigi, Yi Cao, Sudhanva Gurumurthi, Charles Recchia, Andrew Walton, and Vilas Sridharan. A systematic study of ddr4 dram faults in the field. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 991–1002. IEEE, 2023.
26. The Linux Foundation. Open neural network exchange: The open standard for machine learning interoperability. https://onnx.ai/, 2019.
27. Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019.
28. N. Leveson and J. Thomas. An STPA primer. *Cambridge, MA*, 2013.
29. Xugui Zhou, Bulbul Ahmed, James H Aylor, Philip Asare, and Homa Alemzadeh. Data-driven design of context-aware monitors for hazard prediction in artificial pancreas systems. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 484–496. IEEE, 2021.
30. Bin Nie, Lishan Yang, Adwait Jog, and Evgenia Smirni. Fault site pruning for practical reliability analysis of gpgpu applications. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 749–761, 2018.
31. Harald Schafer, Eder Santana, Andrew Haden, and Riccardo Biasini. A commute in data: The comma2k19 dataset. *arXiv preprint arXiv:1812.05752*, 2018.
32. Lishan Yang, Bin Nie, Adwait Jog, and Evgenia Smirni. Enabling software resilience in GPGPU applications via partial thread protection. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*, pages 1248–1259, 2021.
33. Lishan Yang, Bin Nie, Adwait Jog, and Evgenia Smirni. Practical resilience analysis of GPGPU applications in the presence of single- and multi-bit faults. *IEEE Trans. Computers*, 70(1):30–44, 2021.
34. Saurabh Jha, Subho S. Banerjee, Timothy Tsai, Siva Kumar Sastry Hari, Michael B. Sullivan, Zbigniew T. Kalbarczyk, Stephen W. Keckler, and Ravishankar K. Iyer. Ml-based fault injection for autonomous vehicles: A case for bayesian fault injection. In *49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2019, Portland, OR, USA, June 24-27, 2019*, pages 112–124, 2019.
35. Matt Osborne, Richard Hawkins, and John McDermid. Analysing the safety of decision-making in autonomous systems. In *International Conference on Computer Safety, Reliability, and Security*, pages 3–16. Springer, 2022.
36. Francesco Terrosi, Lorenzo Strigini, and Andrea Bondavalli. Impact of machine learning on safety monitors. In *International Conference on Computer Safety, Reliability, and Security*, pages 129–143. Springer, 2022.
37. Xugui Zhou, Anna Schmedding, Haotian Ren, Lishan Yang, Philip Schowitz, Evgenia Smirni, and Homa Alemzadeh. Strategic safety-critical attacks against an advanced driver assistance system. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 79–87. IEEE, 2022.
38. Xugui Zhou, Anqi Chen, Maxfield Kouzel, Haotian Ren, Morgan McCarty, Cristina Nita-Rotaru, and Homa Alemzadeh. Runtime stealthy perception attacks against dnn-based adaptive cruise control systems. *arXiv preprint arXiv: 2307.08939*, 2024.