Enhancing Autonomous Vehicle Safety through N-version Machine Learning Systems

Qiang Wen[1,*], Júlio Mendonça[2], Fumio Machida[1] and Marcus Völp[2]

[1]Department of Computer Science, University of Tsukuba, 305-8573, Japan
[2]Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg, L-1855, Luxembourg

Abstract

Unreliable outputs of machine learning (ML) models are a significant concern, particularly for safety-critical applications such as autonomous driving. ML models are susceptible to out-of-distribution samples, distribution shifts, hardware transient faults, and even malicious attacks. To address the concerns, the N-version ML system gives a general solution to enhance the reliability of ML system outputs by employing diversification on ML models and their inputs. However, the existing studies of N-version ML systems mainly focused on classification errors and did not consider their impacts in a practical application scenario. In this paper, we investigate the applicability of N-version ML approach in an autonomous vehicle (AV) scenario within the AV simulator CARLA. We deploy two-version and three-version perception systems in an AV implemented in CARLA, using healthy ML models and compromised ML models, which are generated using fault-injection techniques and analyze the behavior of the AV in the simulator. Our findings reveal the critical impacts of compromised models on AV collision rates and show the potential of three-version perception systems in mitigating the risk. Our three-version perception system improves driving safety by tolerating one compromised model and delaying collisions when having at least one healthy model.

Keywords

autonomous driving, fault injection, machine learning system, N-version programming, perception

1. Introduction

Rapid machine learning (ML) advancements have led to widespread applications across various domains. ML-based intelligent software systems, including face recognition, medical diagnosis, and autonomous robots, have become integral parts of our daily lives [1, 2]. However, ML models cannot guarantee a correct output in the application context due to ML models' uncertainties in dealing with real samples [3]. Additionally, transient faults (e.g., leading to bit-flip errors [4]) and malicious attacks such as adversarial attacks [5] may affect the system's capability to provide correct outputs, especially when a single ML model is in the software stack [6, 7]. When ML-based applications are incorporated into safety-critical systems, incorrect outputs can cause undesirable consequences. For example, the misrecognition of traffic signs by ML-based classifiers could result in accidents in autonomous driving scenarios. By using this example, we should agree that ensuring the correctness of ML-based system outputs has become a critical concern, especially for systems in safety-critical domains.

Various approaches have been proposed to enhance the robustness of ML systems. ML testing is one of these approaches that focuses on detecting differences between existing and required behaviors of machine learning systems [8]. However, the existing works mainly focus on offline testing rather than runtime monitoring. To improve correctness during runtime, additional safety mechanisms such as data validation [9], safety monitors [10], and redundant architecture [11, 12] must be deployed. Current ML data validation techniques pose operational challenges, including an abundance of false positive warnings and the necessity for manual adjustments. Similarly, safety monitors, while crucial, lack adaptability due to their simultaneous training with the ML model. Model enhancement and specialization led to the generation of large Deep Neural Networks (DNNs), capable of modeling more complex patterns and data relationships, which, consequently, present improved results as output [13]. However, large DNNs should require more computational resources to be executed, which specific systems, such as autonomous vehicles (AVs), may not afford as it would incur extra resource costs (e.g., energy). Although feasible and sometimes suitable, adopting a large DNN in limited-resource systems would incur the use of a single DNN, offering the system a single point of failure, which could cause malfunction of the entire system in the case of hardware or software failures or malicious attacks.

In contrast, adopting redundant architectures offers a more straightforward approach by utilizing diverse ML models and data inputs. Using multiple and diverse ML models, an ML-based system can avoid a single point

The IJCAI-24 Workshop on Artificial Intelligence Safety (AISafety 2024), August 04, 2024, Jeju, South Korea
*Corresponding author.
wen.qiang@sd.cs.tsukuba.ac.jp (Q. Wen); julio.mendonca@uni.lu (J. Mendonça); machida@cs.tsukuba.ac.jp (F. Machida); marcus.voelp@uni.lu (M. Völp)

of failure since replicated models can execute the same tasks, masking failures or misclassifications. Also, adopting model diversity can help the system mitigate problems such as overfitting and adversarial attacks, as different models could have a distinct structure and training data. Leveraging the idea of a traditional software fault tolerant technique, N-version programming (NVP) [14], the *N*-version ML system approach uses *replication* and *diversification* to improve the output reliability of ML systems [12]. By integrating multiple, independently functioning ML models, the N-version system is designed to maintain operation and accurate decision-making even when one or more components are compromised or faulty. The multiple versions of ML models and input data sources are used to generate multiple inference results, which may differ from each other. These results are subsequently analyzed using decision logic (e.g., a voter employing a majority voting rule [15]) or a protocol to agree on a single value (e.g., consensus protocols [16]) to determine the final output. This approach enables the system to detect and mitigate incorrect outputs arising from individual ML models. More recent studies have analyzed the adoption of N-version ML systems and presented their benefits for output reliability [11, 17, 15, 18]. However, none of these works have examined the safety impact in a practical application scenario.

Therefore, this paper leverages N-version ML system architectures for the perception module of AVs, aiming to investigate the impact of such architectures on the safety of autonomous driving scenarios using the CARLA simulator [19]. Specifically, we consider two-version and three-version perception systems, each comprising two or three independent ML modules, respectively, for object detection tasks in AVs. We incorporate multiple versions of ML models within the systems by deploying different versions of the YOLOv5 model. In addition, to simulate failures and errors, caused by transient faults or malicious attacks, we create compromised ML models using the fault-injection tool PyTorchFI [20]. The tool intentionally changes ML model parameters, which can introduce errors into the ML models, representing situations where ML systems may be affected by different types of faults (e.g., radiation, induced memory corruption). Then, we combine healthy and compromised ML models, following an N-version system architecture, and deploy it into an AV running on the CARLA simulator. The results show that single compromised models can significantly impact the AV collision rate in up to 90% of the analyzed scenarios. We also find that the three-version system has the potential to tolerate one compromised model efficiently and delay collisions caused by incorrect object detection when having at least one healthy model. We make the following contributions in this paper:

- We propose the application of N-version ML systems to the perception module of an AV to enhance autonomous driving safety.
- We conduct fault injection experiments to reveal the impact of compromised ML models in the perception module on the safety of AVs simulated in CARLA.
- Through the experiment, we demonstrate the enhanced driving safety achieved by a three-version perception system that can mitigate incorrect outputs from compromised ML models and delay possible AV collisions.

The remainder of the paper is organized as follows. Section 2 presents background and related work. Section 3 details the system and fault model adopted in this work. Section 4 clarifies the research questions addressed in the following experiment and describes the experiment settings. Section 5 discusses the achieved results, focusing on answering the defined research questions. Finally, Section 6 concludes the paper and briefly presents future work.

2. Background and Related Works

2.1. N-version Machine Learning

N-version ML architecture, based on NVP, comprises N ($\geq$2) diverse versions of ML components operating in parallel for the same task [12]. The ML components generate multiple inference results individually, and the final output can be determined using a voting mechanism. Unlike ensemble learning [21], which aims to build a better model by combining weak learners, the N-version ML architecture is configured with pre-trained black-box models and designed for ML system operation. Recent studies have investigated N-version ML approaches to improve system reliability. Xu et al. [22] proposed the NV-DNN, a framework aimed at enhancing the fault tolerance of deep learning systems comprising N independently developed models and decision-making procedures. NV-DNN assumes processing a single input at a time, whereas N-version ML can also consider different inputs to exploit input diversity. Furthermore, diversifying input data can contribute to improving the reliability of N-version ML systems, as demonstrated in works from Machida and Wen [11, 15, 23]. Hong et al. [24] proposed a multimodal deep-learning approach to improve the classification accuracy of remote-sensing imagery, outperforming single-model or single-modality approaches. Mendonça et al. [18] investigated the improvement of output reliability in perception systems through a modeling approach when integrating N-version programming with rejuvenation techniques. Nevertheless, none of the existing studies have shown the effectiveness of the N-version

ML approach in AV safety against the risk of faulty ML models.

2.2. Fault-injection for ML Models

Fault injection is a testing technique used to analyze systems under the presence of faults [25]. This method entails intentionally introducing faults behavior into a system to examine its function under abnormal conditions. The objective is to evaluate whether the system can tolerate faults and continue to operate correctly or will misbehave. Recent studies have investigated fault injection techniques in deep neural networks (DNNs). For example, single bias attack and Gradient descent attack are two types of fault injection attacks proposed to misclassify a specified input pattern into an adversarial class by modifying the parameters used in DNNs by Liu et al. [26]. Tools such as PyTorchFI [20] have been proposed for disturbing DNNs on the PyTorch platform, allowing users to induce perturbations in the weights or neurons of DNNs at runtime. Piazzesi et al. [27] used fault-injection tools to evaluate autonomous agents under the presence of artificial faults and attacks. In this study, we leverage a fault injection tool to evaluate an N-version perception system for AV.

2.3. Autonomous Driving Simulation

CARLA [19] is a well-known and adopted open-source simulator designed for autonomous driving research. The simulation platform supports flexible specification of sensor suites and environmental conditions. CARLA has been extensively used to assess various aspects of autonomous driving. For instance, simulations enable the verification of whether a driving system, trained using data from a simulator, can be effectively deployed on a real car [28]. Besides, works developed by Gao et al. [29] and Piazzesi et al. [27] leverage CARLA to develop and evaluate object detection algorithms tailored for autonomous driving applications. By utilizing the simulation environment, the detection models can be tested under various conditions. In this work, we shall focus on object detection tasks within the perception system, particularly in analyzing N-version architectures for perception systems running in the CARLA simulator.

3. Fault and System Model

We focus on an ML-based perception module running in an AV. A perception module is an essential component of AVs. The perception module leverages inputs from advanced sensors present in an AV. It serves as the vehicle's sensory hub, collecting and processing vast amounts of data to create a detailed understanding of its surroundings. It can integrate inputs from cameras, LiDAR, radar, and ultrasonic sensors, each contributing unique capabilities for detecting and classifying objects such as other vehicles, pedestrians, and road signs, as well as identifying lane markings and traffic signals [30]. The comprehensive sensory data is then forwarded to the planning and prediction modules to form a dynamic 3D map of the environment, enabling the AV to navigate safely and efficiently.

Perception modules heavily rely on ML models to detect obstacles, pedestrians, traffic signs, signals, lanes, and other vehicles from the input captured by cameras and other sensors. Therefore, a failure or simple misclassification of objects in the environment may impact the safe driving behavior of the AV, which may lead to dangerous traffic situations and cause accidents.

Next, we detail the fault model adopted in this work and then present an N-version perception system for AVs, which aims to mitigate the impact of faulty and compromised ML models to enhance AV safety driving.

3.1. Fault Model

The fault model focuses on transient faults and malicious attacks related to ML models' output correctness. Thus, we assume sensors produce correct data and they, as well as other components outside the perception system, are not subject to failures or attacks. On the other hand, we consider vulnerabilities in deep learning frameworks (e.g., PyTorch, TensorFlow, or Caffe) could allow attackers to (1) launch denial-of-service attacks, (2) crash deep learning applications due to memory exhaustion, (3) generate wrong classification outputs by corrupting the classifier's memory, or (4) hijack the control flow to remote control the deep learning application hosting system [31]. The latest CVE reports on Tensorflow (CVE-2023-27506, CVE-2023-25668), PyTorch (CVE-2022-45907), and Caffe (CVE-2021-39158) confirm the presence of such vulnerabilities. Besides, ML models shall be subject to transient faults such as radiation, which are capable of causing bit-flips.

In this way, we assume an ML model can have three possible states: healthy (H), compromised but operational (C), or non-operational (N). When in a healthy state (H), the ML model performs normally, but it intrinsically includes producing incorrect outputs according to its accuracy. When faults or malicious attacks (e.g., radiation, induced memory corruption) affect the ML model, it may cause errors, which could lead to a subsequent failure. When faults or attacks cause errors in the ML model, it reaches a compromised but functional state (C). Compromised ML models can still perform object detection tasks but have a reduced probability of producing correct perception outputs. However, when the errors lead to failures, the ML model completely stops, entering a non-

operational state (N), incapable of executing perception tasks.

In this work, we shall focus on how vulnerability exploitation and fault effects could be mitigated by using N-version ML models. In this way, we assume that errors or failures could harm all (or none) *N* ML models at the same time. This allows us to generalize N-version architecture to consider situations where ML models are executed isolated (e.g., in different cores) and are not subject to the same failures and situations where ML models are affected equally by a single failure. In practice, we demonstrate how artificially injected faults affect distinct ML models' output differently by generating different compromised ML models, as well as the overall effect when the system only has different compromised ML models executing.

3.2. An N-version Perception System

To mitigate the impact of failed and compromised ML models on AV safety, we present an N-version perception system for enhancing perception outputs. Figure 1 shows the architecture of an N-version perception system. We assume a perception system of an AV composed of *N* ML models capable of executing object detection tasks, which aim to avoid AV collisions. We shall focus in this work on situations where data input variation is not employed. It means that all *N* ML models should receive the same data input from the AV sensors (e.g., cameras) to perform object detection. Note that in some systems, it is also possible that different sensor data could be combined through a sensor fusion component before being forwarded to the ML models [32]. After executing the object detection task, each model shall forward its output to a voter, which decides the final perception output based on a pre-defined voting rule. In the adopted system, we consider the voter implementing a majority-based voting rule for simplicity, while other rules can be implemented later. Besides, we assume the voter is implemented in a trustworthy component and shall not be susceptible to malicious attacks or faults. Such mechanism implementation has been demonstrated in practice by previous works, such as Gouveia et al. [33].

Assuming the mentioned architecture, an *N*-version ML perception system can be represented in a set of reachable states S in which $(h, c, n) \in S$ and *h*, *c*, and *n* represent the numbers of ML models in the healthy, compromised, and non-operational state, respectively. Additionally, we assume the voter can automatically detect when an ML model is in a non-operational state (N). Usually, failure detection tools can be easily adopted to verify whether a component is operational. This would be necessary to prevent the voter from waiting indefinitely for the output of non-operational models and for it to be able to reconfigure itself with different pre-determined voting rules automatically.

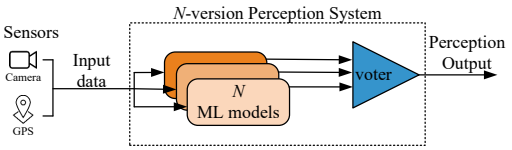


Figure 1: An example of an *N*-version perception system architecture, containing *N* ML models. ML models can assume one of the states in a given moment: healthy (H), compromised but operational (C), or non-operational (N).

4. Experiments

4.1. Objective

The objective of this study is to investigate the applicability of an N-version perception system architecture for AV safety. We aim to answer the following research questions throughout experiments using CARLA.

RQ1: How does a compromised ML model of a perception system impact AV driving safety?

RQ2: How efficiently can an N-version perception system (N=2,3) tolerate compromised and non-operational models?

To address *RQ1*, we set up a simulation environment deploying different compromised ML models into the AV perception system to evaluate its driving behavior. We simulate compromised ML models using PyTorchFI to generate artificial faults in the ML models. We also compare the driving behavior of the compromised ML models against the healthy ML models. To answer *RQ2*, we implement two-version and three-version perception systems, incorporating various combinations of healthy and compromised models. Then, we investigate the driving behavior across all the possible configurations, including entirely healthy, mixed (healthy and compromised), and entirely compromised models. This analysis allows us to evaluate how the N-version ML approach influences the driving behavior of the AV's perception system under various conditions.

4.2. Testbed Setup

We utilize the CARLA AV simulator and a cooperative driving co-simulation framework OpenCDA [34] to simulate a single-lane driving scenario. During the simulation process in OpenCDA, sensors installed on each AV collect the surrounding environment as well as the ego vehicle

information (e.g., 3D LiDAR points and Global Navigation Satellite System (GNSS) data). The collected sensors' data are used by the perception and localization systems for object detection and localization. Subsequently, the perception output, including object 3D pose and ego position, is delivered to the downstream planning system to generate the AV trajectory and, consequently, update the AV's acceleration, speed, and wheel turning. Finally, the planned trajectory and commands are passed to the control system, which generates the final control commands. In this paper, we choose *Town03* in CARLA as the map shown in Figure 2. The yellow oval marks the starting point, and the yellow star marks the endpoint of the simulation run that an AV must execute. Towards this path, the AV relies on its perception system to accurately detect other vehicles and road obstacles.

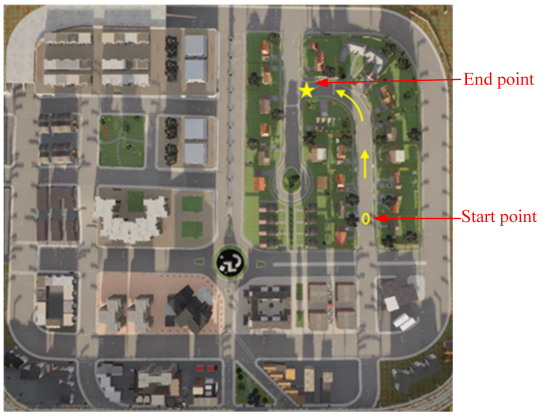


Figure 2: Adopted scenario in Town03 of the CARLA simulator.

Next, we define two-version and three-version systems using different object detection models in the architecture. We employ unmodified versions of the YOLOv5 model [35], including YOLOv5s6, YOLOv5m6, and YOLOv5l6 as healthy models to deploy them into the AV perception system. Then, we generate compromised versions of these models using PyTorchFI [20]. More specifically, adopting PyTorchFI's runtime perturbation feature for weights and neurons in DNNs. Those functionalities are crucial for simulating real-world scenarios where models may encounter unexpected disruptions. Thus, we employ PyTorchFI's *random_weight_inj* function with a weight range of (-100, 300) to mimic the conditions compromised models may encounter. The injection function randomly alters parameters within a randomly selected layer of the neural network, thereby introducing variability into the YOLO image detection algorithm. The degree to which the ego vehicle's perception is impacted (i.e., whether it causes an error) depends on the sensibility of the model layer for the randomly injected weight. After injecting artificial faults in the healthy models, we generate new models, renaming them to YOLOv5s6_FI, YOLOv5m6_FI, and YOLOv5l6_FI. Each one of these models shall represent the ML models when in a compromised state (C). Note that the weight perturbations injected on these compromised models affect all input data (e.g., image frames) during the entire period.

In our three-version perception system, we employ a majority voting rule. When three models are operational (i.e., in the states H or C), the voter provides a perception output when 2-out-of-3 models agree on the same output. The agreement is defined based on the criterion that the bounding boxes (bboxes) have an Intersection over Union (IoU) exceeding 0.8, and the labels are identical. When the majority cannot be reached, the voter provides no perception output. Consequently, the AV does not update its driving properties (e.g., speed and acceleration). When one model stops completely (i.e., entering a non-operational (N) state), the system degrades to a 2-version. In those cases, the voting rule implemented in the voter is that the two models must agree on the same output. Otherwise, it should not provide any perception output. Therefore, the AV can only update its trajectory and driving properties if the voter receives equal output from the two models.

Evaluation Metric: We measure the *collision rate* of the AV as the number of collision frames over the total number of frames in a run. We also give the first collision frame number and total frame number as evaluation metrics. The metrics measure the driving behavior of the AV under different system configurations that a three-version system could assume. We conduct ten runs for each system configuration and report the average of the metrics.

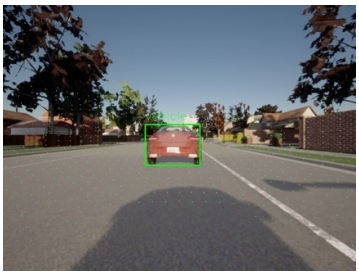
5. Results

In this section, we discuss the experiment results to answer research questions related to the N-version perception systems. We focus on the impact of compromised models and the effectiveness of an N-version perception system to answer the defined research questions.

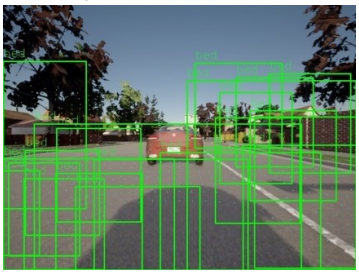
5.1. Evaluation of Compromised ML Models

First, we investigate the effects of compromised models on object detection with an AV adopting a single-version perception system. We compare the object detection results between ML models in the healthy (i.e., in state *H*) and compromised (i.e., in state *C*) states. Recall that an ML model in a non-operational state (i.e., state *N*) cannot produce any output. Thus, the AV cannot afford a single-version architecture with only one model in this

state. Figure 3 illustrates an object detection case of (a) a healthy YOLOv5m model, which accurately detects the vehicle in front, whereas (b) its fault-injected variant, YOLOv5m_FI, produces numerous erroneous bounding boxes, which has more probability of leading the AV to potential collisions.



(a) Using a healthy YOLOv5m model.



(b) Using a compromised YOLOv5m_FI model.

Figure 3: Example of object detection using a healthy and compromised model during AV driving.

Table 1 presents the average values for three metrics across ten runs, including the first collision frame, total number of frames, and collision rate. System state represents (h, c, n), where h, c, and n represent the numbers of ML models in the healthy, compromised, and non-operational state, respectively. The upper part displays the results for single-version perception systems. Notably, the healthy models consistently exhibited a 0% average collision rate, while the compromised models showed significantly higher average collision rates (more than 70%). The AV had a collision in 90% runs when driving with different versions of compromised models. The numbers demonstrated that the AV using compromised models tends to experience collisions from the very onset of the simulation. Figure 4 illustrates the collision and no collision cases in the simulated scenario. In our study, we primarily focus on vehicle-to-vehicle collisions. Even though there is a curve in the lane, other collisions such as vehicle-to-infrastructure collisions will not happen.
Answer to RQ1: Compromised models of an AV perception system demonstrate a high average collision rate of more than 70%, adversely affecting AV driving safety.

(a) No collision case.

(b) Collision case.

Figure 4: Fault injection effect example on the AV for the scenario adopted.

5.2. Evaluation of N-version Perception Systems

Next, we present the results achieved when adopting two- and three-version perception systems. The middle part of Table 1 presents the experimental results for two-version systems. The results indicate that no collisions occurred in all runs when both healthy models were executed (i.e., state (2,0,1)). In contrast, configurations (1,1,1) and (0,2,1) experienced collisions. Notably, the number of collisions in the (1,1,1) configurations was lower than in the (0,1,2) configurations, suggesting that a two-version system with one compromised model can still mitigate some collisions. The average collision rates for the AV under the system state (1,1,1) and (0,2,1) were more than 50%. Additionally, the results demonstrate that the first collision frame, when considering compromised models, was at around frame 64, which is at a very early stage of the simulation. It is an important observation related to the layout of the vehicles during the simulation scenario, in which the ego vehicle starts the simulation in movement and is relatively close to the vehicle in front of it. When two ML models disagree, and the detection of the vehicle in front is abnormal, the ego vehicle tends to have a rear collision with the vehicle in front of it. Thus, when adopting a two-version perception system, the AV would have a short time before entering a critical erroneous state, generating wrong object detection outputs after having at least one ML model compromised.

The bottom part of Table 1 presents the computed results for the three-version perception system. When the system had the majority of models in a healthy state (i.e., (3,0,0) and (2,1,0)), no collisions were observed.

Table 1
Collision data of the experiments over different states in a single, two, and three-version system.

System state	YOLO Model	1st collision frame	Total frames	Collision rate%	# Collisions
Single-version					
(1,0,2)	v5s	NA	687	0	0/10
(1,0,2)	v5m	NA	685	0	0/10
(1,0,2)	v5l	NA	682	0	0/10
(0,1,2)	v5s_FI	119	628	71.40	9/10
(0,1,2)	v5m_FI	103	622	74.33	9/10
(0,1,2)	v5l_FI	89	644	76.72	9/10
Two-version					
(2,0,1)	v5s,v5m	NA	685	0	0/10
(1,1,1)	v5s,v5m_FI	64	704	54.63	6/10
(1,1,1)	v5l,v5m_FI	66	690	63.27	7/10
(0,2,1)	v5s_FI,v5m_FI	64	667	81.22	9/10
Three-version					
(3,0,0)	v5s, v5m, v5l	NA	682	0	0/10
(2,1,0)	v5s, v5m, v5m_FI	NA	693	0	0/10
(2,1,0)	v5s, v5m, v5s_FI	NA	682	0	0/10
(1,2,0)	v5s, v5s_FI, v5m_FI	272	666	28.82	5/10
(1,2,0)	v5m, v5s_FI, v5m_FI	335	654	33.08	7/10
(0,3,0)	v5s_FI, v5m_FI, v5l_FI	187	643	57.00	8/10

The result indicates that a three-version perception system can effectively tolerate at least one compromised model and mask its failures when adopting the majority voting rule. For the configuration (1,2,0), where a majority of the models were compromised, collisions occurred in most runs. However, there were instances where the system successfully avoided collisions. The average collision rates for this configuration were about 30%, significantly lower than those of single-version compromised models. This observation suggests that even a system with more compromised models has the potential to prevent collisions under certain circumstances. Notably, the average first collision frame in the (1,2,0) configurations was much later compared to single-version compromised models. This is a significant observation as it demonstrates that the system can delay the onset of erroneous outputs. This delay could provide critical additional time for the AV to take evasive action, thereby possibly avoiding a collision. Finally, for the configuration (0,3,0), where all models were compromised, most runs ended in collisions, as expected due to the lack of healthy models to correct the errors. However, two out of ten runs did not result in a collision, suggesting that even with all models compromised, specific conditions within the scenario might prevent failures temporarily.

Answer to RQ2: A three-version perception system can efficiently tolerate one compromised model. Even when the majority of the models are compromised, the system has the potential to prevent some collisions or delay erroneous perception outputs.

5.3. Discussion

The findings from three-version perception systems demonstrate the application of the N-version ML approach in improving the safety of AV. Specifically, the configurations did not result in a collision when most models were in healthy states, showing the three-version perception system's ability to mitigate disruptions. Although collisions still occur in some configurations with the majority of compromised models, the system shows the potential to delay erroneous perception outputs that could lead to collisions. This capability is crucial in environments where even a minor delay in failure onset can provide essential time for initiating corrective actions, thereby preventing potentially catastrophic outcomes.

Limitations. In this experiment, we did not consider the cost and performance overhead imposed by the redundant modules. The use of multiple ML models in the N-version perception system introduces additional computational overheads and may be costly to implement in a real vehicle. The overhead and cost can be mitigated by adjusting the number of modules activated and/or the frame rates. The perception output can also be enhanced by diversifying the input data without using multiple models [36]. Such a design optimization under resource constraints needs to be investigated further in

future work. Besides, our current evaluation is limited to a short run on one specific map. Further experiments with more diverse driving scenarios are needed to make more general conclusions.

6. Conclusions and Future Work

In this study, we explored the practical application of N-version ML system architectures through experiments conducted in the autonomous vehicle simulator CARLA. By deploying two-version and three-version perception systems for object detection tasks, we investigated the effectiveness of incorporating multiple versions of both healthy and compromised ML models within the systems. Our findings demonstrate that compromised models within the perception system significantly impact the AV collision rate, with rates exceeding 70%. In addition, we observed that three-version perception systems have the potential to mitigate object detection misclassifications, tolerating one compromised model and delaying collisions when at least one healthy model remains operational. In future work, we consider evaluating other system architectures and exploring alternative decision-making mechanisms beyond simple majority voting rules that could improve N-version ML systems' output correctness.

Acknowledgments

This work was supported by JST SPRING Grant Number JPMJSP2124, and partly supported by JSPS KAKENHI Grant Numbers 19K24337 and 22K17871. This work has also been supported by the German Research Council (DFG) and by the Luxembourg Fond Nationale de Recherche (FNR) through the Core Inter Project ByzRT (C19-IS-13691843).

References

[1] J. A. Sidey-Gibbons, C. J. Sidey-Gibbons, Machine learning in medicine: a practical introduction, BMC medical research methodology 19 (2019) 1–18.

[2] H. J. Vishnukumar, B. Butting, C. Müller, E. Sax, Machine learning and deep neural network—artificial intelligence core for lab and real-world test and validation for adas and autonomous vehicles: Ai for efficient and quality test and validation, in: Intelligent systems conference (IntelliSys), 2017, pp. 714–721.

[3] M. Henne, A. Schwaiger, G. Weiss, Managing uncertainty of ai-based perception for autonomous systems, in: AISafety@ IJCAI, 2019, pp. 11–12.

[4] M. A. Hanif, F. Khalid, R. V. W. Putra, S. Rehman, M. Shafique, Robust machine learning systems: Reliability and security for deep neural networks, in: International Symposium on On-Line Testing And Robust System Design, 2018.

[5] S. Qiu, Q. Liu, S. Zhou, C. Wu, Review of artificial intelligence adversarial attack and defense technologies, Applied Sciences 9 (2019) 909.

[6] A. Toschi, M. Sanic, J. Leng, Q. Chen, C. Wang, M. Guo, Characterizing perception module performance and robustness in production-scale autonomous driving system, in: IFIP International Conference on Network and Parallel Computing, Springer, 2019, pp. 235–247.

[7] Apollo perception, 2019. URL: http://www.fzb.me/apollo/specs/perception_apollo_5.0.html.

[8] J. M. Zhang, M. Harman, L. Ma, Y. Liu, Machine learning testing: Survey, landscapes and horizons, IEEE Transactions on Software Engineering (2020).

[9] W. Wu, H. Xu, S. Zhong, M. Lyu, I. King, Deep validation: Toward detecting real-world corner cases for deep neural networks, in: Proc. of the 49th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2019, pp. 125–137.

[10] R. S. Ferreira, J. Arlat, J. Guiochet, H. Waselynck, Benchmarking safety monitors for image classifiers with machine learning, in: Proc. of IEEE Pacific Rim International Symposium on Dependable Computing (PRDC), 2021, pp. 7–16.

[11] F. Machida, On the diversity of machine learning models for system reliability, in: IEEE Pacific Rim Int'l Symp. on Dependable Computing (PRDC), 2019, pp. 276–285.

[12] F. Machida, N-version machine learning models for safety critical systems, in: Proc. of the DSN Workshop on Dependable and Secure Machine Learning, 2019, pp. 48–51.

[13] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, nature 521 (2015) 436–444.

[14] L. Chen, A. Avizienis, N-version programming: A fault-tolerance approach to reliability of software operation, in: Proc. of 8th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-8), 1978, pp. 3–9.

[15] Q. Wen, F. Machida, Reliability models and analysis for triple-model with triple-input machine learning systems, in: Proc. of the 5th IEEE Conference on Dependable and Secure Computing, 2022, pp. 1–8.

[16] R. Olfati-Saber, J. A. Fax, R. M. Murray, Consensus and cooperation in networked multi-agent systems, Proceedings of the IEEE 95 (2007) 215–233.

[17] S. Latifi, B. Zamirai, S. Mahlke, Polygraphmr, enhancing the reliability and dependability of cnns, in: Proc. of 50th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2020, pp. 99–112.

[18] J. Mendonça, F. Machida, M. Völp, Enhancing the reliability of perception systems using n-version programming and rejuvenation, in: Proc. of the 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), 2023, pp. 149–156.

[19] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, V. Koltun, Carla: An open urban driving simulator, in: Proc. of the 1st Annual Conference on Robot Learning, 2017, pp. 1–16.

[20] A. Mahmoud, N. Aggarwal, A. Nobbe, J. R. S. Vicarte, S. V. Adve, C. W. Fletcher, I. Frosio, S. K. S. Hari, Pytorchfi: A runtime perturbation tool for dnns, in: 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), 2020, pp. 25–31.

[21] Z.-H. Zhou, Ensemble methods: foundations and algorithms, CRC press, 2012.

[22] H. Xu, Z. Chen, W. Wu, Z. Jin, S. Kuo, M. R. Lyu, Nv-dnn: towards fault-tolerant dnn systems with n-version programming, in: Proc. of the 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), 2019, pp. 44–47.

[23] Q. Wen, F. Machida, Characterizing reliability of three-version traffic sign classifier system through diversity metrics, in: Proc. of the 34th International Symposium on Software Reliability Engineering (ISSRE), 2023, pp. 333–343.

[24] D. Hong, L. Gao, N. Yokoya, J. Yao, J. Chanussot, Q. Du, B. Zhang, More diverse means better: Multimodal deep learning meets remote-sensing imagery classification, IEEE Transactions on Geoscience and Remote Sensing 59 (2020) 4340–4354.

[25] M. C. Hsueh, T. K. Tsai, R. K. Iyer, Fault injection techniques and tools, Computer 30 (1997) 75–82.

[26] Y. Liu, L. Wei, B. Luo, Q. Xu, Fault injection attack on deep neural network, in: Proc. of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2017, pp. 131–138.

[27] N. Piazzesi, M. Hong, A. Ceccarelli, Attack and fault injection in self-driving agents on the carla simulator – experience report, in: Computer Safety, Reliability, and Security: 40th International Conference, SAFECOMP 2021, Springer-Verlag, York, UK, 2021, pp. 210–225.

[28] B. Osiński, A. Jakubowski, P. Zięcina, P. Miłoś, C. Galias, S. Homoceanu, H. Michalewski, Simulation-based reinforcement learning for real-world autonomous driving, in: IEEE international conference on robotics and automation (ICRA), 2020, pp. 6411–6418.

[29] W. Gao, J. Tang, T. Wang, An object detection research method based on carla simulation, Journal of Physics: Conference Series 1948 (2021) 012163.

[30] S. D. Pendleton, H. Andersen, X. Du, X. Shen, M. Meghjani, Y. H. Eng, D. Rus, M. H. Ang, Perception, planning, control, and coordination for autonomous vehicles, Machines 5 (2017) 6.

[31] Q. Xiao, K. Li, D. Zhang, W. Xu, Security risks in deep learning implementations, in: IEEE Security and Privacy Workshops (SPW), 2018, pp. 123–128.

[32] R. Maurice, M. Gerla, Autonomous driving: Sensor fusion for multiple sensor types, in: Proceedings of the IEEE International Conference on Intelligent Transportation Systems, IEEE, 2012.

[33] I. P. Gouveia, M. Völp, P. Esteves-Verissimo, Behind the last line of defense: Surviving soc faults and intrusions, Computers & Security 123 (2022) 102920. doi:10.1016/j.cose.2022.102920.

[34] R. Xu, H. Xiang, X. Han, X. Xia, Z. Meng, C.-J. Chen, C. Correa-Jullian, J. Ma, The opencda open-source ecosystem for cooperative driving automation research, IEEE Transactions on Intelligent Vehicles 8 (2023) 2698–2711. doi:10.1109/TIV.2023.3244948.

[35] G. Jocher, et al., ultralytics/yolov5: v5.0 - yolov5-p6 1280 models, aws, supervise.ly and youtube integrations, 2021. URL: https://doi.org/10.5281/zenodo.4679653.

[36] K. Wakigami, F. Machida, T. Phung-Duc, Reliability and performance evaluation of two-input machine learning systems, in: 2023 IEEE 28th Pacific Rim International Symposium on Dependable Computing (PRDC), IEEE, 2023, pp. 278–286.