Kayotee: A Fault Injection-based System to Assess the Safety and Reliability of Autonomous Vehicles to Faults and Errors

Saurabh Jha*, Timothy Tsai†, Siva Hari†, Michael Sullivan†, Zbigniew Kalbarczyk*,
Stephen W. Keckler†, and Ravishankar K. Iyer*
*University of Illinois at Urbana-Champaign, Urbana-Champaign, IL, USA, 61801
†Nvidia Corporation, Santa Clara, CA, USA, 94086

***Abstract*—Fully autonomous vehicles (AVs), i.e., AVs with autonomy level 5, are expected to dominate road transportation in the near-future and contribute trillions of dollars to the global economy. The general public, government organizations, and manufacturers all have significant concern regarding resiliency and safety standards of the autonomous driving system (ADS) of AVs . In this work, we proposed and developed (a) 'Kayotee' - a fault injection-based tool to systematically inject faults into software and hardware components of the ADS to assess the safety and reliability of AVs to faults and errors, and (b) an ontology model to characterize errors and safety violations impacting reliability and safety of AVs. Kayotee is capable of characterizing fault propagation and resiliency at different levels - (a) hardware, (b) software, (c) vehicle dynamics, and (d) traffic resilience. We used Kayotee to study a proprietary ADS technology built by Nvidia corporation and are currently applying Kayotee to other open-source ADS systems.**

I. Introduction

The safety and reliability of autonomous vehicles (AVs) are significant concerns among all the stakeholders. Our previous work [1] characterized a California Department of Motor Vehicles (DMV) dataset on reported AV road testing and showed that as many as 36% of disengagements were caused by computer system problems and 64% were due to machine learning problems. AV research has traditionally focused on improving machine learning and artificial intelligence models. However, as these models are deployed at large scale on computing platforms, the focus is to assess the resilience and safety features of the compute stack driving the AVs. The effects of faults and errors in the hardware (GPUs, CPUs and other processing units) running the AV software stack is not well understood. Recent work [2]–[5] exclusively focus on the resiliency of deep neural networks (DNNs) to hardware faults and errors without accounting for the inherent resiliency in the software stack. [6], [7] study the safety of AVs by injecting sensor-related permanent faults such as Gaussian noise, occlusion, etc. into publicly available autonomous driving system (such as CARLA [8] and Open Pilot [9]). However, these autonomous driving system (ADS) are overly simplistic with few sensors and are not representative of a production ADS. Moreover, such studies have limited scope as they cannot characterize error masking and propagation of transient, and permanent faults in the ADS. We believe our work is the first to study the impact of transient errors, intermittent errors and permanent errors (with some limitations) on AV safety and reliability.

Our work focuses on developing a fault injection tool to assess hardware and software resilience and its implication on the safety of ADS. We developed 'Kayotee' to inject faults into software and hardware components of the ADS in a closed-loop environment to empirically characterize resilience, safety, and the error propagation and masking properties in the ADS system. The fault models were chosen to emulate the representative transient faults in the hardware and software (by corrupting software state variables) to expose error masking limits. The capabilities of Kayotee are (i) injection of hardware and software faults using fault models bundled in the tool, (ii) selection of fault sites based on software components (sensor inputs, object perception, sensor fusion, planner and controller), hardware components (CPU vs. GPU), and machine learning algorithms (DNN vs. non-DNNs), (iii) creation and execution of multiple traffic scenarios, (iv) simulation of the closed-loop environment where parameters (such as speed, acceleration, distance traveled by the AV and actuation command outputs sent by the ADS) follow truncated normal distributions and capture data from multiple modules to compare fault-free run (i.e., golden run) values, i.e., non-injected run outputs to injected run outputs, and (v) calculation of resilience and safety violation parameters.

In comparison to AVFI [6], Kayotee is capable of characterizing error propagation and masking in the ADS using a closed-loop simulation environment and is capable of injecting bit flips directly into GPU and CPU architectural state. Our fault injector was tested on the Nvidia DriveWorks platform, and we plan to extend it to Apollo [10] and CARLA [8], [11].

To help improve the safety and reliability characteristics of ADS, we need to answer the following research questions —

Q1: Which software modules are most vulnerable to faults/errors among different parts of the ADS (such as object perception, path perception, localization and planning)?

Q2: Do errors in the ADS platform contribute statistically more degradation in safety and resilience characteristics than degradation because of inherent data quality and inaccuracies in ML/AI-techniques ?

Q3: Is there any statistical relationship between safety/resilience and input characteristics (such as #vehicles, #people etc.) given errors?

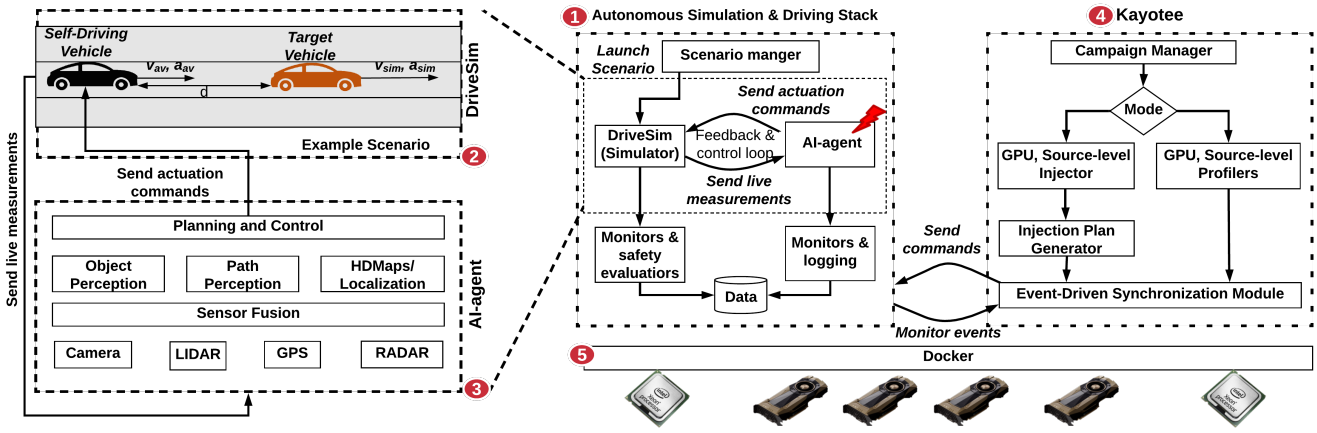


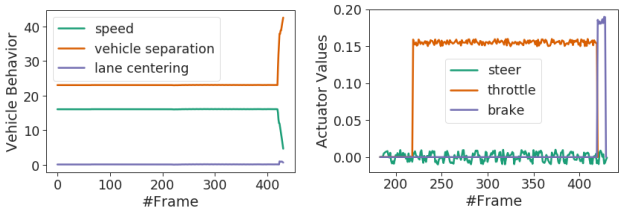
Figure 1: End-to-end safety and reliability evaluation of ADS using Kayotee

Q4: Is there any statistical difference in error susceptibility among different computing platforms - CPUs and GPUs?

Due to proprietary restrictions, we only share fault injection experiment methodology and do not provide any numerical values/results that can identify key-characteristics of the Nvidia driving platform or its susceptibility to faults. The views, opinions, tools and results contained or described in this article are those of the authors and do not necessarily reflect the official policy or position of Nvidia Corporation. However, we plan to share results with the community by implementing our tools and techniques for open-source ADS and simulators such as Apollo [10] and Carla [8].

II. Autonomous Vehicles

An AV is any vehicle that uses an ADS technology capable of supporting and replacing a human driver in the tasks of controlling the main functions of steering, acceleration, and monitoring the surrounding environment (e.g., other vehicles/pedestrians, traffic signals, and road markings) [12]. In this work, we used a proprietary autonomous driving stack (supporting an AI-agent running at automation level 5) and simulation systems (see Fig 1, 1) developed by Nvidia Corporation to showcase the use cases of Kayotee and its usefulness in characterizing the safety and reliability of ADS. Like any other ADS system, the Nvidia ADS consists of the Nvidia AI-agent, marked as 3 in Fig 1 which further consists of five basic modules - (a) sensors and sensor fusion module, (b) object perception, (c) path perception, (d) maps and localization, and (e) planning and control in addition to several safety check mechanisms (not shown in the figure). In addition to testing Nvidia ADS on private/public roads, Nvidia tests the ADS using a custom Unreal Engine-based simulation engine, DriveSim [13] (marked as 2). DriveSim is capable of simulating complex urban scenarios by using a library of urban layouts, buildings, pedestrians, vehicles, and weather conditions (e.g., sunny, rainy, and foggy). An example scenario is shown in 2. In this scenario, an Nvidia AI-agent controlled vehicle and a DriveSim controlled vehicle are placed on highway driving at different speeds (v) and acceleration (a) separated by a distance d along with other highway objects (such as road signs, traffic lights, etc. not shown in the figure). Such parameterization allows mimicking situations such as - (a) a target vehicle slowing down, (b) a stationary target vehicle, (c) an accelerating target vehicle, and many more. The *scenario manager* toolkit in DriveSim can be used to select various pre-created urban scenarios. The *monitoring and safety evaluators* toolkit is used to subscribe to DriveSim measurements providing ground truth values associated with the scenario in realtime and then used to evaluate safety parameters associated with the AI-driven vehicle (such as whether the vehicle is at the center of the lane, whether the vehicle maintains a minimum distance from other vehicles, and whether the speed of the vehicle is within the safety limits with respect to other vehicles and traffic rules). Sample vehicle behavior (speed, lane centering, and vehicle separation) and actuation output measurements are shown in Fig. 2a and Fig. 2b respectively.



(a) Vehicle behavior (b) Actuation values

Figure 2: Simulation results of a single run without fault injection

III. Kayotee Software Architecture

Kayotee was used to profile the ADS workload running on a computing platform (consisting of Intel Xeon CPUs and Nvidia discrete GPUs[1]) and then inject faults using representative fault models (by uniform random sampling of fault locations in the GPU architectural state).

A. Experimental Strategy

We built 'Kayotee' to characterize error propagation and masking (a) in the Nvidia GPUs and CPUs, (b) in the ADS,

[1]In this work, we did not inject faults into the Drive AGX Pegasus [14] platform.

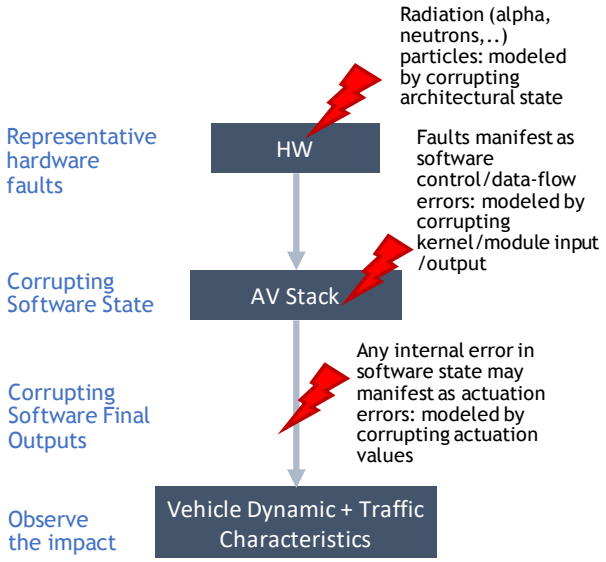


Figure 3: Experimental Strategy

and (c) in vehicle dynamics and traffic. For each of these characterizations, we built a corresponding injector capable of injecting faults such that errors manifest in the corresponding locations. In the case of GPUs, we used the GPU injector to inject architectural-state faults (see Section III-C), and SLI (Section III-D) to inject into the inputs and outputs of the ADS kernels (or modules). Corrupting the final output (actuation commands) of the ADS sent to the AV helps us measure resilience associated with vehicle dynamics and traffic. As shown in Figure 3, low-level circuit-, micro-architectural-, and RTL-faults manifest as architectural-state faults (injected through GPU-injector). The architectural-state faults that do not get masked manifest as errors in the internal state of the kernels of software stack and any error that does not get masked in the kernel propagates to the output of the kernel. Finally, errors that are not masked before the point of sending actuation commands lead to incorrect actuation commands (actuation errors) sent to the AV. Thus, our approach aids the measurement of fault masking and propagation at different levels and its corresponding impact on the safety of the AV.

B. Kayotee

Kayotee is a fault injection tool that can inject transient (a) hardware faults, and (b) software state errors. Kayotee is bundled with a campaign manager that takes an XML configuration file as input to select a fault model, software or hardware module sites for fault injection, the number of faults, and a scenario. The campaign manager uses the specified configuration to (a) profile the ADS workload, (b) generate a fault plan, and (c) inject one or more transient faults per run into the ADS system (until required confidence levels are reached). To monitor and understand the impact of transient faults on the safety and reliability of the ADS, it is important to ensure that the simulation is deterministic. However, a control-loop based system by definition is non-deterministic in nature. Thus, we developed an 'event-driven synchronization' module that coordinated between all the toolkits (DriveSim, monitoring, evaluators, and AI-agent) to ensure that the parameters of the moving objects (AI-vehicle, other vehicles, etc.) in the scenario (including the AI-vehicle) roughly follow a truncated normal distribution.

C. GPU Injector Fault Models

We consider transient faults in the functional units (e.g., arithmetic and logic units, and load store units), latches, and unprotected SRAM structures of the GPU processor. Such transient faults are modeled by injecting bit-flips (single and double) in the outputs of the executing instructions. If the destination register is a general-purpose register or a condition code, one bit (or two bits) is randomly selected to be flipped. For store instructions, we flip a randomly selected bit(s) in the stored value. Since we inject errors directly into live state (destination registers), our error model does not account for various masking factors in the lower layers of the hardware stack such as circuit-, gate-, and micro-architecture-level masking as well as masking due to errors in architecturally untouched values. The GPU injection tool uses the same profiling and fault-injection plan generation mechanism as used in SASSIFI [15]. We do not consider faults in cache, memory, and register files because we assume that they are protected by ECC.

D. SLI Fault Models

The goal of SLI (Source-Level Injection) is to corrupt the internal state (by corrupting output variables) of the ADS software components. In Table I, we show some of the variables from each of the ADS modules (see 3 in Figure 1) that were targeted using SLI. The fault models supported by SLI are as follows –

- *Random*: The chosen variable is randomly modified to a value within the range of zero to vehicle speed limit on the road (e.g., 65mph for majority of US urban interstate roads). For example, in case of object_class we use possible object classes supported in the Nvidia ADS but for pid_measured_values we choose a random value between zero and 65mph (highway speed limit).
- *Fixed*: The chosen variable is always set to the fixed value. It helps to evaluate the worst possible fault cases, e.g., pid_output for the speed controller is always set to the maximum supported value. This fault-model is useful to inject known fault conditions, and most importantly when generating worse-case intermittent or permanent errors to understand the maximum resiliency offered by the ADS.
- *Scale*: The chosen variable is scaled to some ratio of the current value of the variable.
- *Disappear*: The chosen output is either not delivered to the next module, if it does not lead to the software crash, or set to null (or zero) if it does lead to software crash.

E. Error and Safety Metrics

To characterize error and safety in the ADS, we propose a new ontology model to capture a range of issues that can occur in the real-world and also observed from fault injection campaigns. An ontology model capturing the fault manifestation in the ADS is shown in Fig. 4. Any run with an injected fault is labeled as *activated* if any of the monitored values (such as object classification, bounding box, actuator command values, vehicle measurements, etc.) do not fall within the

ADS Module	Output Variables (fault injection target)	Fault Model
Path perception	lane_type, lane-coordinates	random, fixed, scale, disappear
Object perception	num_detected_objects, object_class, object_coordinates	
Planning & control	pid_measured_value, pid_target_value, pid_output	
Sensor outputs	camera frames	Gaussian noise, occlusion (from [6])

Table I: SLI supported fault models. Only select few output variables are shown in the table.

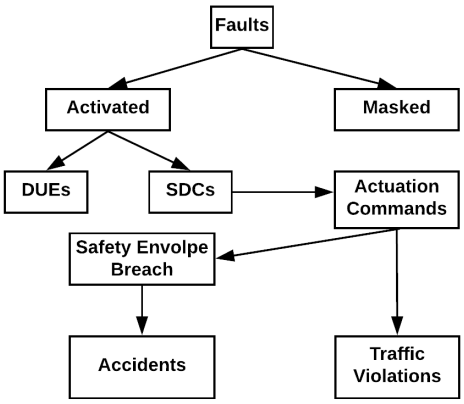


Figure 4: Ontology model for fault manifestation in AVs

expected value range obtained in the golden runs, i.e., a run with no injection or if leads to a hang or crash. We use an IQR (interquartile range)-based outlier detection algorithm [16] combined with the range of the distribution in the golden run to label a variable as containing an erroneous value. Such error labeling is scenario-specific as the ground truth values are obtained by running the ADS for a specific scenario with no injection; otherwise, it is labeled as *masked*. An activated fault can be further classified as *DUE* (detectable uncorrectable error, such as a hang or crash) or *SDC* (silent data corruption). We label a run as SDC if there is a change in the value of any of the monitored variables in the AV stack compared to the golden run value. SDCs can lead to actuation errors, and any such run is labeled as *actuation-error*. Actuation errors can lead to a breach in the safety envelope or traffic violations or both. A breach in the safety envelope can lead to an accident.

In our study, we defined following scenario-agnostic safety metrics for breaches of safety:

Safety envelope breach occurs when the collision distance (i.e., which is the distance traveled by the vehicle from its current position to collision point) between AV and any other object on the road (moving or stationary) is less than the stopping distance (refer to section 2.6.1 of [17]). The collision point can be calculated using trajectory estimation approaches [18]. The stopping distance (D_s) is given by

$$D_s = D_p + D_r + D_b \quad (1)$$

Perception distance (D_p) is the distance the vehicle travels in ideal conditions from the time that the driver (human or AI-agent) of the vehicle sees a hazard until the brain or ADS recognizes it. The average perception time for an alert human driver is 1.75 seconds [17]. For an AI-agent the worst-case recognition time is $1/FPS$, where FPS is the frames per second processed by the AI-agent.

Recognition distance (D_r) is the distance the vehicle will continue to travel in ideal conditions before physically hitting the brakes in response to a hazard seen ahead. The average human driver has a reaction time of 0.75 to 1.0 seconds [17]. For an AI-agent, this corresponds to the time taken to send an actuation command after recognizing the hazard.

Braking distance (D_b) is the distance the vehicle will travel in ideal conditions while braking. On a highway at 24.5872 meters/sec (55mph), a vehicle will travel a minimum of 64 meters of braking distance [17]. If two vehicles are on a highway traveling in the same direction, the vehicle separation may only be 20 meters, but the collision distance is significantly higher due to vehicle dynamics. The braking distance depends on the road surface and the type, weight, speed, and acceleration of the vehicle. In this paper, we only considered speed to calculate braking distance. However, the braking distance can be more accurately calculated using previously proposed models [19], [20].

Lane centering breach occurs when the distance from the lane center changes by more than 0.5 meters.

IV. Conclusion and Future Work

In this work we presented Kayotee, a fault injection tool, along with methodologies to empirically assess the fault propagation, resilience, and safety characteristics of the ADS. The main objective of Kayotee is to evaluate the effects of the faults on the ADS and to investigate the maximum number of the faults that the ADS can tolerate before a safety violation occurs. Although Kayotee may help identify some of the software design issues or bugs in the ADS, it does not systematically evaluate the software design or bugs (e.g., using static checkers).

The future work involves —

- porting Kayotee to the publicly available open-source Apollo [10] system and to share the resilience/safety characteristics with the community,
- beam-testing of the CPUs and GPUs to understand the differences in emulated faults on CPUs/GPUs to representative faults that may occur in the field.

References

[1] 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 *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018.

[2] 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.

[3] K. Pei, Y. Cao, J. Yang, and S. Jana, "DeepXplore: Automated whitebox testing of deep learning systems," in *Proc. of the 26th Symposium on Operating Systems Principles*, 2017, pp. 1–18.

[4] B. Salami, O. Unsal, and A. Cristal, "On the resilience of rtl nn accelerators: Fault characterization and mitigation," *arXiv preprint arXiv:1806.09679*, 2018.

[5] B. Reagen, U. Gupta, L. Pentecost, P. Whatmough, S. K. Lee, N. Mulholland, D. Brooks, and G.-Y. Wei, "Ares: a framework for quantifying the resilience of deep neural networks," in *Proceedings of the 55th Annual Design Automation Conference*. ACM, 2018, p. 17.

[6] S. Jha, S. S. Banerjee, J. Cyriac, Z. T. Kalbarczyk, and R. K. Iyer, "Avfi: Fault injection for autonomous vehicles," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2018.

[7] A. H. M. Rubaiyat, Y. Qin, and H. Alemzadeh, "Experimental resilience assessment of an open-source driving agent," *arXiv preprint arXiv:1807.06172*, 2018.

[8] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proc. of the 1st Annual Conf. on Robot Learning*, 2017, pp. 1–16.

[9] CommaAI, "OpenPilot: Open Source Driving Agent," https://github.com/commaai/openpilot, Accessed: 2018-09-12.

[10] Baidu, "Apollo Open Platform," http://apollo.auto, Accessed: 2018-09-02.

[11] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," in *Proc. of International Conf. on Robotics and Automation (ICRA)*, 2018.

[12] SAE International, *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*, Sep 2016.

[13] NVIDIA, "NVIDIA Drive Simulation," https://www.nvidia.com/en-us/self-driving-cars/drive-constellation/, Accessed: 2018-09-02.

[14] Nvidia Corporation, "Drive Pegasus," https://www.nvidia.com/en-us/self-driving-cars/drive-platform/, Accessed: 2018-09-12.

[15] 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.

[16] A. M. Committee *et al.*, "Robust statistics–how not to reject outliers. part 1. basic concepts," *Analyst*, vol. 114, no. 12, pp. 1693–1697, 1989.

[17] California DMV, "Driving Safely," https://www.dmv.ca.gov/portal/dmv/?1dmy&urile=wcm:path:/dmv_content_en/dmv/pubs/cdl_htm/sec2, Accessed: 2018-08-05.

[18] M. Brown, J. Funke, S. Erlien, and J. C. Gerdes, "Safe driving envelopes for path tracking in autonomous vehicles," *Control Engineering Practice*, vol. 61, pp. 307–316, 2017.

[19] D. Wu, J. Li, X. Shu, X. Zha, and B. Xu, "Test analysis and theoretical calculation on braking distance of automobile with abs," in *International Conference on Computer and Computing Technologies in Agriculture*. Springer, 2010, pp. 521–527.

[20] J. Yi, L. Alvarez, X. Claeys, and R. Horowitz, "Emergency braking control with an observer-based dynamic tire/road friction model and wheel angular velocity measurement," *Vehicle system dynamics*, vol. 39, no. 2, pp. 81–97, 2003.