Beyond DNN Silo-Testing: Integrating Autonomous System State

Anonymous Author(s)

ABSTRACT

Adversarial testing tends to focus on DNNs in isolation, to the exclusion of the full system state and system behaviors resulting from sequences of DNN output. In this work we propose a more holistic approach to DNN testing that accounts for the effects of perturbations on the system state. Our insight is that, when an adversarial perturbation is situated in the environment and encountered by the system, the way it is sensed and processed by the system depends on system state. Our approach involves three key elements: 1) integration of simulator into the testing process to update state, 2) optimization of perturbation over time, and 3) fitting perturbations to sequences of inputs consistent over spatio-temporally ordered states. These ideas lay a foundation for the testing of systems with DNNs that rely on spatio-temporally related inputs. We illustrate the potential of our idea through adversarial perturbation of the physical environment of an autonomous vehicle. We define a broader research agenda around this more holistic DNN testing approach that accounts for system state.

ACM Reference Format:
Anonymous Author(s). 2022. Beyond DNN Silo-Testing: Integrating Autonomous System State. In *Proceedings of The 44th International Conference on Software Engineering (ICSE 2022).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Autonomous systems are becoming ubiquitous. We live with cars, drones, vacuum cleaners, and warehouse robots, operating with higher and higher levels of autonomy. In spite of their ubiquity, autonomous systems' potential is often hampered by a susceptibility to misbehavior. For example, Tesla self-driving cars have shown a propensity to crash into emergency vehicles when their lights are flashing [14] or to interpret the full moon as a yellow traffic light and slow down without warning [15]. Such events highlight the importance of thoroughly testing these systems. More specifically, since the software supporting such autonomy increasingly relies on deep neural networks (DNNs) to encode behavior that would be very difficult to develop through traditional programming, testing of DNNs is now critical. This is reflected in the effort our community has allocated to such work [1, 3, 6–9, 11, 12, 16, 17, 20, 21, 23–28, 30, 32, 33].

Today, most testing efforts for the DNNs driving these systems operate in a silo, targeting the DNN in isolation without considering dependencies with the full system state. Such approaches are effective at determining the robustness of the network to typical or adversarial contexts [13]. For a DNN generating a steering angle for self-driving vehicle, silo-testing might help to judge the DNN's ability to correctly predict steering angle on images of an unfamiliar road or on images with environmental noise for which it was not explicitly trained. However, it will miss cases in which the evolving state of the system has an unexpected effect on the input of the DNN, leading to sometimes-catastrophic system misbehavior. For example, a vehicle under test may accelerate around a curve at rapidly increasing velocity. Environmental blurring of the input images has been tested and accounted for to not disrupt steering predictions. However, the acceleration causing the blur makes the vehicle incapable of turning sharply without losing control past some velocity threshold.

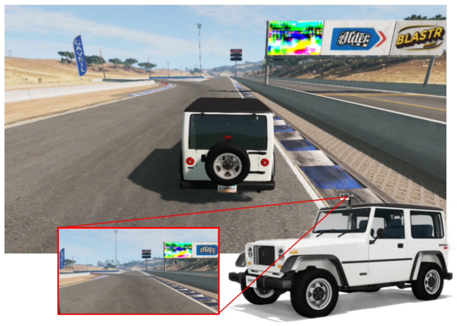


Figure 1: Vehicle driving by a roadside billboard.

In this work we hypothesize and later show that the complex spatial-temporal dependencies of these autonomous systems to their execution environment make existing testing techniques insufficient. Hence, instead of DNN silo-testing, we advocate for testing of DNNs to take into account the system state. This means that the system state must reflect the changes caused by the test execution (i.e., if the DNN generates a steer angle the car must adjust its steering), and it also means that the the next input to the DNN will depend on the updated system state (i.e., the steered car now senses a different image than it would without the steering). The core idea is to include, as part of the testing process, how the DNN output affects the system state and how the system state affects the next round of inputs to the DNN.

Next, we provide a motivating example to illustrate how accounting for system state can dramatically affect testing results. Then, we sketch an approach to show what it would take to incorporate system state into the test generation process. Last, we briefly explore the potential of the proposed approach and find that it can detect more than three times the failures than a state-of-the-art technique, and present a broader research agenda.

2 MOTIVATION

Consider a self-driving car on a road with turns, lane markers, vegetation, and billboards like that in Figure 1. A camera mounted on the vehicle collects images at 30 fps. A DNN then consumes each of those images to produce a sequence of inputs to steer the car [5]. However, such DNNs are prone to adversarial inputs that may cause misbehaviors [2].

One way to attempt to detect misbehaviors like this is through adversarial testing. The state of the art in adversarial image generation takes individual images from a data set, applies some changes that should not affect the DNN output or affect it in known ways, and compares the DNN output for the original and the perturbed image [10]. For example, DeepXplore [18] generates adversarial perturbations using gradient ascent that vary a minimal number of pixels. DeepRoad [31] applies changes that mimic certain weather conditions. DeepHunter [29] aims to make changes that preserve the key features, like changing the image on a billboard would preserve the road in the driving environment.

Such techniques, however, have a fundamental limitation: the tests do not account for how the system functions and the space of possible changes to system state as a result of the DNN input. For example, a real-life malfunction might result in one dead pixel in the image, but that dead pixel will be in a fixed location for the entirety of the car's test run. Thus, adversarial tests that render different adversarial pixels on per-image bases for different images are not physically realistic for the system. Furthermore, a dead pixel that causes a large difference on one image may not have a significant effect on a system as it is corrected by the next captured image or it is attenuated by a constraint imposed by the system state. This makes single-image test error measurements less meaningful in terms of how the perturbation will ultimately affect the system behavior.

DeepBillboard [33] took a first step in accounting for system state changes by considering a sequence of images from the validation set where a car drives past a billboard, similar to Figure 1. DeepBillboard then performs an optimization to generate an adversarial billboard (the perturbation) that maximizes steering loss on all of those images. This produces an adversarial perturbation that can be placed in an environment and be applied to consecutive, physically consistent images. However, this technique has a major weakness: it operates on a **stale state**. That is, the optimization procedure operates on images collected under normal driving, not accounting for the effect of perturbed actuation on the system state in later timesteps.

To enable the capture of system **live-state** for test generation, we propose to embed a simulator into the adversarial generation cycle such that the collection and generation phases are interleaved. Having a simulator in the loop, with additional functions to keep the perturbation consistent over time, we are able to enact the perturbation as we generate it to update the system state, while simultaneously gathering **input constraints** to constrain the effects of the perturbation to physically possible inputs to the system.

Figure 2 illustrates the implications of using a live-state technique to generate adversarial billboards using an input image sequence of length N. The image sequences resulting from the two collection trajectories are shown at the bottom of the figure. The features of the images, such as lane markers, vegetation, obstacles, and vanishing point of the road at time t_0 are similar between techniques, but deviate significantly in later timesteps t_i and t_N reflecting the different system states (different poses which leads to different images being captured). The state-stale technique generates a billboard

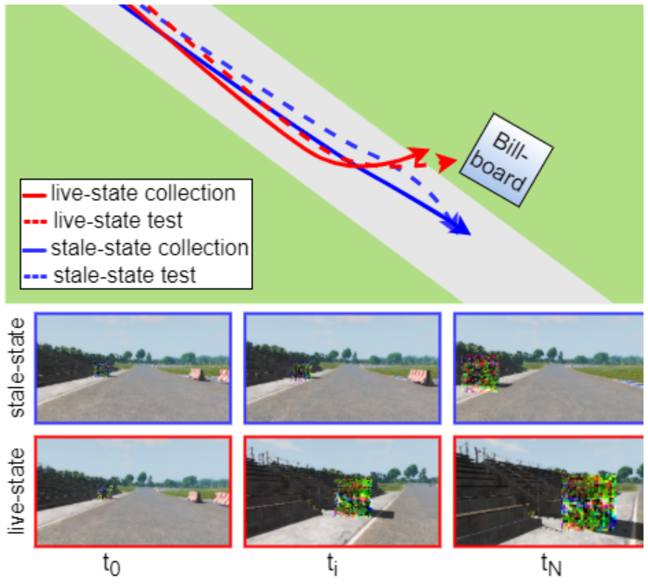


Figure 2: Input images for a stale-state perturbation versus a live-state perturbation and resulting trajectories. Accounting for live-state leads to a crash.

using the input sequence from following a normal trajectory in the collection phase (blue solid line), which causes the test run (blue dotted line) to barely drift from the normal trajectory, as the perturbation stops working on images collected under the new state. The car ultimately returns to a normal trajectory. When incorporating system live-state, state is updated such that the effects of the perturbation are reflected in the system state as seen in the live-state sequence of images outlined in red at the bottom of Figure 2. As a result, the test trajectory (red dotted line) closely mimics the collection trajectory as the vehicle state is kept up-to-date and does not drift out of phase from the collection trajectory.

3 APPROACH

Our goal is to generate perturbations situated in the environment that account for system live-state over space and time in order to cause undesirable behaviors.

Figure 3 provides an overview of the approach applied to the motivating example. The critical components are the Generator of Perturbation over a State Sequence (GPSS) and the simulator. First, the simulator captures the system live-state, including the image sensed by the system and the most recent steering input in the case of the motivating example. Second, this state is added to the sequence of images and input constraints. Third, GPSS uses the updated sequence, DNN, and a goal state, such as "hard left turn", to update the perturbation and pursue that goal state in the next timestep. Fourth, the generated perturbation is injected into the simulator. The process is then repeated, with the simulation stepping forward a timestep using as input θ_t' to produce an updated θ_t informed by the limitations of the system in its current (live) state. The process is repeated until some stopping criteria is reached – in this case, that the billboard is out of view or a crash occurs –

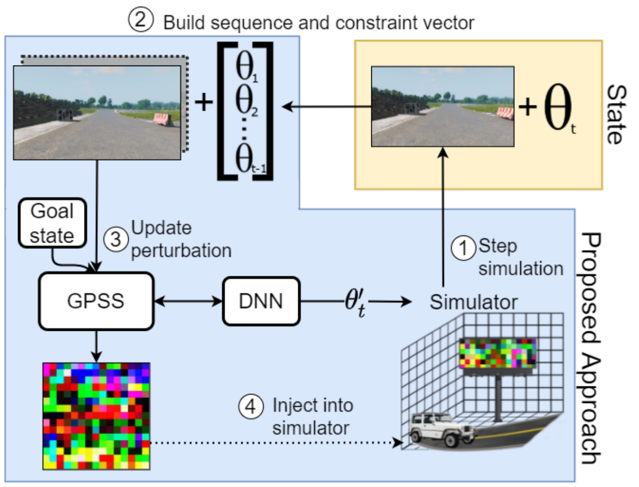


Figure 3: Overview of approach

and a perturbation (final billboard), incorporating constraints from the entire sequence, is obtained and ready for testing.

3.1 Simulator

The simulator is integral to the success of the proposed approach. We assume it has enough knowledge of the system state and the test environment to render relevant state changes. The simulator updates the system at each timestep to maintain a system live-state, consistent with the sensed input from the environment, the DNN commands, and the previous system state.

Note that, given a DNN command, the simulator ensures that **a system input influenced by the perturbation is bounded by the system states reachable from the current live state**. This is advantageous over existing testing approaches that consider the DNN in isolation because the DNN outputs may be curtailed by the system capabilities (e.g., a steering angle can only change so much in a time frame). More formally, at any timestep t in the sequence:

$$\theta_t = system.state.bound(\theta'_t)(\text{where } \theta'_t = DNN(img_t + pert_t)) \tag{1}$$

3.2 GPSS

Generating a perturbation over input sequences must account for two properties. First, **the perturbation must maximize state change toward the goal state**. Given an image *img*, a perturbation *pert*, a DNN that governs the behavior of the system, an input x and the output of the DNN given that input *DNN(x)*, and a loss function $\mathcal{L}$ of the difference in DNN commands engendered by *img+pert*:

$$\underset{pert_t \rightarrow goal}{argmax} \;(\mathcal{L}(DNN(img_t)), (DNN(img_t + pert_t))) \tag{2}$$

Second, **the resulting perturbation must be consistent over time and space**. As the current state of the system depends on the successful early application of perturbation, the perturbation update must maintain the validity of those previous states. Moreover, we do not want to sacrifice the effect of perturbation achieved in previous steps for the sake of maximizing the current perturbation. The input constraint sequence provides a record of how to direct loss when optimizing the current perturbation. Given the final perturbation (the final version of the billboard) $pert_N$, the next equation enforces that the input constraint sequence is maintained such that the resulting input sequence to the DNN stays consistent across all live states of the sequence:

$$\underset{pert_N}{argmin}(\sum_t \mathcal{L}(DNN(img_t + pert_N), DNN(img_t + pert_t))) \tag{3}$$

These properties are enforced jointly at every step of the perturbation generation in relation to previous and subsequent system and environment states through minimization of the loss function.

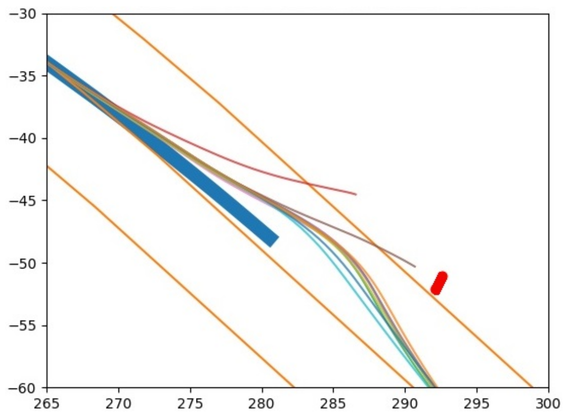
3.3 Implementation for Autonomous Vehicle

For instantiating our approach for an autonomous vehicle, we use a DNN-steered vehicle operating in BeamNG [4], a high-fidelity soft-body physics simulator for realistic driving and vehicle damage. This simulator provides access to the car state, including its steering angle and sensed images. The simulator also enables enacting a perturbation in the environment at each timestep. We can command the simulator to spawn the vehicle on a trajectory near the source of perturbation in a variety of roads and scenarios. The DNN we use takes in an image collected from the simulator to determine the steering input to the car. Those images encode the system state of the car (e.g., how close it is to the edge of the road, whether it is currently navigating a turn).

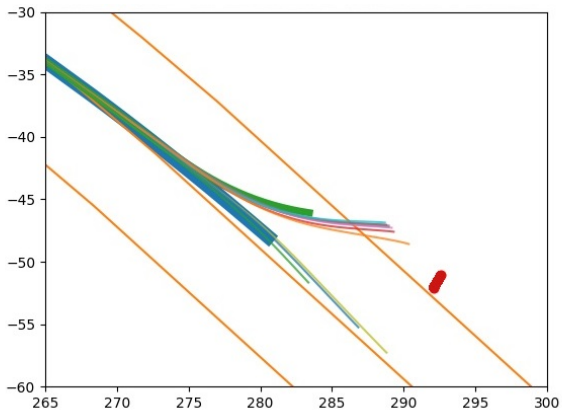
We used a billboard to act as the source of perturbation, *pert*. Here, the delta between next reachable states in Equation 2 is defined in terms of difference in steering angle after perturbation has been applied towards the steering angle goal state. To maximize the state delta as defined in Equation 2 and make an analogous comparison to DeepBillboard, we chose to run the car off the road. This consisted of maximizing steering to the left or right as much as possible at every step. We chose to turn towards the source of perturbation (i.e., the billboard) to allow for the perturbation to stay in sight of the onboard camera longer and give the perturbation a longer window to run the car off the road. Once the vehicle is spawned, the approach collects an image and a steering angle from the simulator to be added to the sequence of inputs, and performs joint loss optimization to enforce a constraint sequence on DNN output satisfying Equations 2 and 3. An input constraint θ_t constrains the steering input of the vehicle according to possible range of actuation at a given system state according to Equation 3. Then, temporary billboard $pert_t$ is injected into the simulated environment, and the simulator is stepped forward by one timestep, updating the state of the vehicle. As the car nears the billboard, it increases in size in the image, and as the orientation of the onboard camera changes in relation to the billboard, the billboard warps to accommodate the change in perspective to satisfy Equation 3. This process is repeated until the onboard camera loses sight of the billboard and our technique produces a billboard with the full sequence as the end product of our approach. To test our approach, we then inject that billboard into the driving environment, and execute test runs with the vehicle starting from the same starting point to compute the crash rate caused by the perturbed billboard.

4 EXPLORING THE APPROACH POTENTIAL

We now begin to explore the potential of the approach to cause system failures in the limited context illustrated in Section 2. More



(a) DeepBillboard.



(b) Proposed Approach.

Figure 4: Effect of billboard perturbation for Deepbillboard and our approach. Road is defined by orange lines, the billboards are in red, the thick blue line is the normal trajectory, and the green thick line is the collection trajectory.

specifically, the testing subject is an autonomous Jeep-like vehicle operating in the BeamNG simulator [4]. It utilizes a DAVE2 network [5, 22] converted to pytorch [19] that consumes camera images to steer the vehicle around a racetrack. As trained, without external influences, the system is able to navigate close to the centerline of a given target race-track indefinitely.

The adversarial testing process aims to expose configurations for a billboard next to the road that would cause a vehicle crash. A reason for exploring this particular context is that it is one of the few that has a state-of-the-art technique for it in Deepbillboard [33]. Deepbillboard and the proposed approach depend on many parameters such as the billboard size and location, the features of the portion of road, the camera resolution and rate, and the different adaptations of the loss function. The proposed approach has even more parameters such as weighting of the sequence. At this stage we have only sampled that space of variables, but among the tens of cases we explored we found that the **proposed approach causes crashes in over 70% of the runs while Deepbillboard is only able to cause crashes in 20% of the runs.**

Figure 4 depicts a typical set of results from DeepBillboard and the proposed approach using the same parameters. The orange lines depict the road, with the axes showing the dimensions of the driving environment in meters. Both approaches to generate the billboard perturbation were designed to pull the car towards a billboard situated on the left-hand side of the road. We executed a set of 10 tests utilizing the billboard from each approach to account for random variation introduced by the simulator. DeepBillboard, Figure 4a, shows a high variance of tests paths, with two of them ending in a crash and the remaining eight returning to the normal trajectory. This confirms our intuition about the impact of not accounting for system state. As the perturbation takes effect and the collection sequence and test sequence diverge, the perturbation becomes less effective and the car returns to the expected trajectory.

Our approach, Figure 4b, has lower variance and greater precision. Seven of the ten test runs ended in a crash into a structure along the left-hand side of the road. These seven runs closely follow the collection trajectory, shown in green. The remaining three test runs return to the normal trajectory. We speculate this is due to noise from the simulator causing the car to miss those early perturbation steps, which in turn causes the image sequence to deviate further and further from the collection sequence and the perturbation to cease to affect the image sequence under test.

5 BROADER RESEARCH AGENDA

This work points to the need for a broader research agenda on testing DNNs that is more holistic in considering the system state. In particular, we believe three directions require more attention.

Technical challenges. The prototyped loss functions are rather primitive. They do not account, for example, for the different influences of state across a sequence (i.e., earlier images should affect the perturbation more than later ones) or for the transfer potential of the pertubation to other contexts (i.e., the same billboard on a different road). They also ignore the cost of the proposed approach. Iteratively capturing the system state and applying gradient ascent at every step slows down test generation by an order of magnitude. Techniques are needed to make the approach more efficient.

Characterizing a Complex Space of Factors. There are many and often confounding factors that may affect the success of our approach. These factors include DNN architecture (i.e., certain layers being more amenable to analysis), system attributes and constraints (i.e., the steering angle depends on the system steer state), scenario attributes (i.e., system initial velocity, straight versus curved road), and perturbation space (i.e., billboard size and placement). This will require extensive empirical studies to draw principles that guide how to best configure the approach parameters to cause crashes.

Beyond Perturbing Billboards to Crash Cars. In the context of autonomous vehicles, there are countless opportunities to inject adversarial perturbations (e.g., license plates, road decals, graffiti). However, the approach is not limited to this type of systems. A drone localizing a target or a robot arm analyzing the orientation of an object both rely on the interpretation of images to accomplish a task. These emerging systems with rich states would be appealing targets for the proposed approach. Furthermore, perturbations do not have to be adversarial; they can be designed to cooperate with the system to, for example, navigate a hairpin curve.

REFERENCES

[1] Aniya Aggarwal, Pranay Lohia, Seema Nagar, Kuntal Dey, and Diptikalyan Saha. 2019. Black Box Fairness Testing of Machine Learning Models. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Tallinn, Estonia) *(ESEC/FSE 2019)*. Association for Computing Machinery, New York, NY, USA, 625–635. https://doi.org/10.1145/3338906.3338937

[2] Naveed Akhtar and Ajmal Mian. 2018. Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Survey. *IEEE Access* 6 (2018), 14410–14430. https://doi.org/10.1109/ACCESS.2018.2807385

[3] Teodora Baluta, Zheng Leong Chua, Kuldeep S. Meel, and Prateek Saxena. 2021. Scalable Quantitative Verification For Deep Neural Networks. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 312–323. https://doi.org/10.1109/ICSE43902.2021.00039

[4] BeamNG. 2020. BeamNG.drive vehicle simulator. https://www.beamng.com/game/.

[5] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. 2016. End to End Learning for Self-Driving Cars. *CoRR* abs/1604.07316 (2016). arXiv:1604.07316 http://arxiv.org/abs/1604.07316

[6] Swaroopa Dola, Matthew B. Dwyer, and Mary Lou Soffa. 2021. Distribution-Aware Testing of Neural Networks Using Generative Models. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 226–237. https://doi.org/10.1109/ICSE43902.2021.00032

[7] Alessio Gambi, Tri Huynh, and Gordon Fraser. 2019. Generating Effective Test Cases for Self-Driving Cars from Police Reports. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Tallinn, Estonia) *(ESEC/FSE 2019)*. Association for Computing Machinery, New York, NY, USA, 257–267. https://doi.org/10.1145/3338906.3338942

[8] Xiang Gao, Ripon K. Saha, Mukul R. Prasad, and Abhik Roychoudhury. 2020. Fuzz testing based data augmentation to improve robustness of deep neural networks. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 1147–1158. https://doi.org/10.1145/3377811.3380415

[9] Simos Gerasimou, Hasan Ferit Eniser, Alper Sen, and Alper Çakan. 2020. Importance-driven deep learning system testing. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 702–713. https://doi.org/10.1145/3377811.3380391

[10] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6572

[11] Divya Gopinath, Hayes Converse, Corina S. Păsăreanu, and Ankur Taly. 2019. Property Inference for Deep Neural Networks. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering* (San Diego, California) *(ASE '19)*. IEEE Press, 797–809. https://doi.org/10.1109/ASE.2019.00079

[12] Fabrice Harel-Canada, Lingxiao Wang, Muhammad Ali Gulzar, Quanquan Gu, and Miryung Kim. 2020. Is Neuron Coverage a Meaningful Measure for Testing Deep Neural Networks?. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 851–862. https://doi.org/10.1145/3368089.3409754

[13] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. 2020. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Computer Science Review* 37 (2020), 100270.

[14] Chris Isidore and Peter Valdes-Dapena. [n.d.]. Tesla is under investigation because its cars keep hitting emergency vehicles. *CNN* ([n. d.]). https://www.cnn.com/2021/08/16/business/tesla-autopilot-federal-safety-probe/index.html

[15] Tim Levin. [n.d.]. Tesla's Full Self-Driving tech keeps getting fooled by the moon, billboards, and Burger King signs. *Business Insider* ([n. d.]). https://www.businessinsider.com/tesla-fsd-full-self-driving-traffic-light-fooled-moon-video-2021-7

[16] Yuanchun Li, Jiayi Hua, Haoyu Wang, Chunyang Chen, and Yunxin Liu. 2021. DeepPayload: Black-box Backdoor Attack on Deep Learning Models through Neural Payload Injection. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 263–274. https://doi.org/10.1109/ICSE43902.2021.00035

[17] Brandon Paulsen, Jingbo Wang, and Chao Wang. 2020. ReluDiff: differential verification of deep neural networks. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 714–726. https://doi.org/10.1145/3377811.3380337

[18] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2019. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. *Commun. ACM* 62, 11 (Oct. 2019), 137–145. https://doi.org/10.1145/3361566

[19] pytorch. 2019. PyTorch. https://pytorch.org/.

[20] David Shriver, Sebastian Elbaum, and Matthew B. Dwyer. 2021. Reducing DNN Properties to Enable Falsification with Adversarial Attacks. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 275–287. https://doi.org/10.1109/ICSE43902.2021.00036

[21] Andrea Stocco, Michael Weiss, Marco Calzana, and Paolo Tonella. 2020. Misbehaviour prediction for autonomous driving systems. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 359–371. https://doi.org/10.1145/3377811.3380353

[22] tech-rules. 2017. DAVE2-Keras. https://github.com/tech-rules/DAVE2-Keras.

[23] Yuchi Tian, Ziyuan Zhong, Vicente Ordonez, Gail E. Kaiser, and Baishakhi Ray. 2020. Testing DNN image classifiers for confusion & bias errors. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 1122–1134. https://doi.org/10.1145/3377811.3380400

[24] Huiyan Wang, Jingwei Xu, Chang Xu, Xiaoxing Ma, and Jian Lu. 2020. Dissector: input validation for deep learning applications by crossing-layer dissection. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 727–738. https://doi.org/10.1145/3377811.3380379

[25] Jingyi Wang, Jialuo Chen, Youcheng Sun, Xingjun Ma, Dongxia Wang, Jun Sun, and Peng Cheng. 2021. RobOT: Robustness-Oriented Testing for Deep Learning Systems. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 300–311. https://doi.org/10.1109/ICSE43902.2021.00038

[26] Jingyi Wang, Guoliang Dong, Jun Sun, Xinyu Wang, and Peixin Zhang. 2019. Adversarial sample detection for deep neural network through model mutation testing. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, Joanne M. Atlee, Tevfik Bultan, and Jon Whittle (Eds.). IEEE / ACM, 1245–1256. https://doi.org/10.1109/ICSE.2019.00126

[27] Zan Wang, Hanmo You, Junjie Chen, Yingyi Zhang, Xuyuan Dong, and Wenbin Zhang. 2021. Prioritizing Test Inputs for Deep Neural Networks via Mutation Analysis. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 397–409. https://doi.org/10.1109/ICSE43902.2021.00046

[28] Mohammad Wardat, Wei Le, and Hridesh Rajan. 2021. DeepLocalize: Fault Localization for Deep Neural Networks. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 251–262. https://doi.org/10.1109/ICSE43902.2021.00034

[29] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. 2019. DeepHunter: A Coverage-Guided Fuzz Testing Framework for Deep Neural Networks. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Beijing, China) *(ISSTA 2019)*. Association for Computing Machinery, New York, NY, USA, 146–157. https://doi.org/10.1145/3293882.3330579

[30] Shenao Yan, Guanhong Tao, Xuwei Liu, Juan Zhai, Shiqing Ma, Lei Xu, and Xiangyu Zhang. 2020. Correlations between Deep Neural Network Model Coverage Criteria and Model Quality. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 775–787. https://doi.org/10.1145/3368089.3409671

[31] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. DeepRoad: GAN-Based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems. In *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 132–142. https://doi.org/10.1145/3238147.3238187

[32] Peixin Zhang, Jingyi Wang, Jun Sun, Guoliang Dong, Xinyu Wang, Xingen Wang, Jin Song Dong, and Ting Dai. 2020. White-box fairness testing through adversarial sampling. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 949–960. https://doi.org/10.1145/3377811.3380331

[33] Husheng Zhou, Wei Li, Zelun Kong, Junfeng Guo, Yuqun Zhang, Bei Yu, Lingming Zhang, and Cong Liu. 2020. DeepBillboard: Systematic Physical-World Testing of Autonomous Driving Systems. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. 347–358.