Finding Property Violations through Network Falsification: Challenges, Adaptations and Lessons Learned from OpenPilot

Meriel von Stein and Sebastian Elbaum

ABSTRACT

Openpilot is an open source system to assist drivers by providing features like automated lane centering and adaptive cruise control. Like most systems for autonomous vehicles, Openpilot relies on a sophisticated deep neural network (DNN) to provide its functionality, one that is susceptible to safety property violations that can lead to crashes. To uncover such potential violations before deployment, we investigate the use of falsification, a form of directed testing that analyzes a DNN to generate an input that will cause a safety property violation. Specifically, we explore the application of a state-of-the-art falsifier to the DNN used in OpenPilot, which reflects recent trends in network design. Our investigation reveals the challenges in applying such falsifiers to real-world DNNs, conveys our engineering efforts to overcome such challenges, and showcases the potential of falsifiers to detect property violations and provide meaningful counterexamples. Finally, we summarize the lessons learned as well as the pending challenges for falsifiers to realize their potential on systems like OpenPilot.

KEYWORDS

testing, autonomous systems, adversarial examples, falsification, formal methods, neural nets

ACM Reference Format:

Meriel von Stein and Sebastian Elbaum. 2022. Finding Property Violations through Network Falsification: Challenges, Adaptations and Lessons Learned from OpenPilot . In *Proceedings of The 37th IEEE/ACM International Conference on Automated Software Engineering (ASE 2022).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In 2016, comma.ai unveiled OpenPilot, one of the first open-source systems for self-driving vehicles. OpenPilot enables vehicles with Automated Lane Centering, Adaptive Cruise Control, Forward Collision Warning, and Lane Departure Warning to assist drivers in limited contexts, prioritizing highway driving [9]. Since its inception, OpenPilot has grown to support over 150 commercial vehicles and its userbase has grown substantially, with over 2,750 new users a week by the end of 2020 [6].

OpenPilot autonomy relies heavily on the `supercombo` DNN, an end-to-end driving model for making predictions about various features of the driving environment, including: optimal paths over upcoming timesteps, position and probability of lead cars, locations of lane lines and road edges, and the probability of OpenPilot disengagement, braking, and acceleration over upcoming timesteps. Model inputs are taken from camera sensors, human input to the system, system configuration, and recurrent system state. `supercombo` predictions are used by many different parts of the OpenPilot software, such as lateral (turning) and longitudinal (gas/brake) planning and identifying driving events such as `laneChange` or `roadCameraError`.

The functionality of OpenPilot depends on images that can be reliably interpreted by the `supercombo` network. To maintain safe operation, the network output must conform to certain safety properties. For example, one such property might specify that adding a low level of noise to a camera image input should not change `supercombo` lane line predictions by more than $\frac{1}{4}$ of a standard lane width.

To test these properties, DNN falsification aims to uncover violations related to the robustness of DNN predictions. Falsification techniques extend adversarial example generation approaches to find violations to safety properties within some measure of local robustness [1–5, 7, 13, 14]. Falsifiers take in a network and a safety property and attempt to find a violation of that property, resulting in either a counterexample or a timeout. For the previous property associated with lane line prediction, a counterexample would consist of an original image plus minimal noise that causes `supercombo` to predict lane lines that deviate by over $\frac{1}{4}$ of a lane width from predictions on the original image.

Recent studies involving falsifiers illustrate their potential to uncover counterexamples for valuable safety properties [12]. However, the falsified networks used in these studies do not capture the complexity we encounter in DNNs used in autonomous vehicles such as `supercombo`, as well as other real-world applications of DNNs at scale. Moreover, recent complex networks often involve multi-dimensional inputs and outputs with inherent dependencies that require more sophisticated properties than early falsifiers can represent. In this work we:

- Analyze the challenges in applying a state-of-the-art falsification framework, DNNF [12], to a state-of-the-art DNN for autonomous vehicles, `supercombo`. Our analysis reveals that recent DNNs introduce conceptual issues beyond scale and format that challenge today's falsifiers, such as multiple inputs, multi-dimensional outputs, and complex operators and architecture.
- Extend DNNF to enable its application to a set of safety properties of `supercombo` and perform a study illustrating the potential for falsification to uncover meaningful counterexamples that will lead to violations.
- Collect lessons learned from our analysis, application, and tool extension, as well as a recognition of the challenges that remain for falsification to be more broadly deployed on DNNs used in autonomous vehicles.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ASE 2022, October 10 - 14, 2022, Ann Arbor, MI, USA

© 2022 Association for Computing Machinery.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

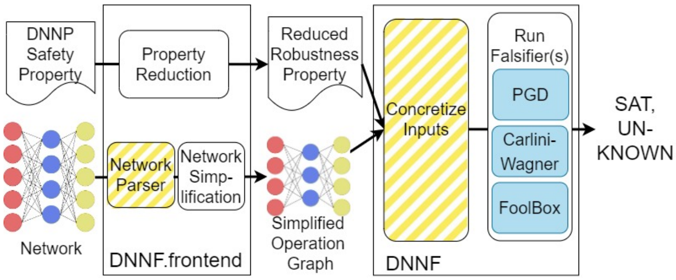


Figure 1: Approach Overview. Extended functionality is shown in yellow stripes.

2 OBSERVATIONS AND CHALLENGES IN FALSIFYING `SUPERCOMBO`

The `supercombo` model is a large-scale, multi-input network with a relatively complex architecture. The architecture of `supercombo` consists of a series of 70 convolutional layers, followed by a block of 23 layers with multiple connections between each layer. `supercombo` takes in four input tensors: `input_imgs`, `desire`, `traffic_convention`, and `initial_state`, all of which enter at different points in the network. Only `input_imgs` is processed by the convolutions at the beginning of the network. The `input_imgs` tensor is a composite of two reordered image tensors collected consecutively by an onboard camera. The network outputs a single tensor with shape=(1, 6472). The output provides information on many aspects of the driving environment, including traffic car locations, path waypoints over upcoming timesteps, and probabilities associated with certain predictions.

Among state-of-the-art falsifiers, Shriver et al. have developed the framework DNNF [12] that enables the use of many existing adversarial example generators to falsify more sophisticated safety properties. DNNF architecture is shown in Figure 1. The DNNF frontend consumes a safety property written using property specification language DNNP and a network, and outputs a simplified network in an intermediate representation *operation graph* and a robustness property that is equivalid to the safety property. These outputs are then consumed by DNNF, which concretizes inputs that are not being manipulated, according to the property specification. The input that can be manipulated is referred to as *symbolic input*. DNNF then runs a falsification technique, converting the operation graph according to the deep learning (DL) framework utilized by the falsifier, and manipulating the symbolic input to falsify the safety property. In this way, DNNF functions as a framework to allow existing falsification techniques to run on more sophisticated properties and a wider variety of networks regardless of the DL framework in which they were implemented.

Early falsifiers [7, 14] did not leverage the internal structure of the DNN, instead fuzzing it as a black box. Treating the DNN as a black box makes them applicable to any DNN independent of internal complexity, but they are significantly slower at finding deeper violations because they lack an understanding of the DNN structure. Moreover, though these early falsifiers could be extended to support networks with complex relationships between inputs and/or outputs, they currently do not, in part because of the conventions of falsification benchmark networks. Subsequent falsifiers [3, 5, 13] used adversarial attacks, enabling these techniques to work on any type of network. However, these falsifiers were limited to only local robustness properties, instead of falsifying specific properties written across inputs and outputs using first-order logic.

	Benchmark Networks[12]	Supercombo Network
Total Operators	74	356
Unique Operators	11	15
Unique Activation Functions	2	4
Inputs	1	4
Input Dimensionality	(200,200,1)	(1,12,128,256) (1,8) (1,2) (1,512)
Outputs	1	1
Output Dimensionality	(1,10)	(1,6472)

Table 1: Comparison of the maximum values found for complexity aspects of DNNF benchmark artifacts versus the `supercombo` network.

Applying DNNF to a network like `supercombo` is not possible out-of-the-box due to assumptions about the networks it consumes. Four benchmarks containing 52 networks in total were used to benchmark DNNF. Each benchmark contains trained models categorized by application: ACAS-Xu, DNNF-CIFAR-EQ, DNNF-GHPR, and Neurify-DAVE. Table 1 shows the differences between the largest networks used to benchmark DNNF and `supercombo` compared in the ONNX learning framework [8]. The first row shows the maximum number of operators[1] per benchmark network versus `supercombo`, with `supercombo` containing almost 5 times as many. `supercombo` contains 4 more unique operators and 2 more unique activation functions than the most varied networks in the DNNF benchmarks. Significantly, `supercombo` takes 4 inputs whereas all benchmark networks take 1, with the largest dimensions of those inputs being a 1-channel 200×200 image. All networks produce one output, but the dimensions of those outputs are vastly different. The largest output dimensions in the benchmark are (1,10) compared to supercombo's (1, 6472).

The differences in Table 1 highlight the types of networks for which DNNF was designed. First, multi-input networks are not supported, despite many network architectures requiring multiple inputs. For example, recurrent neural networks are multi-input networks that take in previous output from the network to supplement new input. `supercombo` is itself a recurrent network, taking in its own previous state as an input. Similarly, long-short term memory (LSTM) networks are a specific kind of recurrent network for timeseries data, and have appeared in real-world self-driving vehicle software [11]. The falsification step of DNNF presents further challenges for multi-input networks. Falsification relies on manipulating symbolic input in order to falsify properties. So, multi-input networks may need to restrict which inputs are used for falsification, while keeping others constant.

Second, DNNF needs to be able to convert all operators in a network into an intermediate representation. If operator types are not represented internally by DNNF, the network cannot be consumed

[1] ONNX operators [8] are a weighted function or matrix operation such as Add, Gemm, or Conv2D, or an activation function. Unlike neurons, operators consider weighted functions or matrix operations and activation functions separately.

for falsification. More recent large-scale networks involve operators that are not supported by the reduction process. For example, supercombo involves the Split operator that is more common in large networks to direct the flow of information. In total, 20 unique operators are present across all networks in this set of benchmarks. However, the size of the individual networks means that some operators endemic to large-scale networks are not present.

Third, the scale and complexity of benchmark network output stands in contrast to that of supercombo. The largest benchmark output dimensionality is (1,10), which comes from ResNet networks' classification score output. Although these scores are correlated, they are low-dimensional in that a limited number of correlations can be inferred between them. In contrast, supercombo output consists of far more features with high correlation, such as the predicted path and anticipated acceleration, or locations of lead vehicles and lane line predictions. Falsifying for one set of outputs may have unintended consequences on many others. Moreover, many of these individual pieces of information have dimensionality in the x-y-z plane and over time, further complicating the relationships between predicted features. The interconnected and high-dimensional nature of supercombo output requires complex properties with low overhead to falsify.

3 ADAPTING DNNF FOR SUPERCOMBO

Given these challenges, we explore extending DNNF to make falsification of supercombo viable in order to produce valuable counterexamples of safety properties.

We enacted the following workflow to extend DNNF and determine relevant properties for supercombo: (1) Extending the DNNF frontend to handle new operators in DL frameworks PyTorch and Tensorflow to support parsing of the input network; and (2) Extending DNNF to support multiple inputs, and the specification of a subset of them as constants.

Figure 1 displays the extended portions of DNNF and DNNF frontend in diagonal yellow stripes. The DNNF frontend takes in a network and parses it recursively to create an intermediate representation. This requires for all operators in the network to be analyzed, determining the shape of the input and output for each operator, and tracing them throughout the analysis. This is performed in the network parser by a visitor pattern visiting each operator in the network and converting it into a DL framework-agnostic representation to form the operation graph as an intermediate representation of the network.

For supercombo, only one operator needed to be added to the DNNF frontend, alongside extensions to existing operators. The Split operator splits the input tensor into a set of chunks of parameterized size along a parameterized dimension. The output of this operator is those "chunks" of the original tensor. It is a means to reroute the flow of information in a network. With the introduction of Split and the accompanying re-routing of output information from Split nodes, the OutputSelect operator had to be parameterized to the number of input and output channels and internal logic extended. The OutputSelect operator is endemic to the DNNF frontend to assist the operation graph visitor in directing output from a preceding operator with multiple outputs to subsequent operators. While the OutputSelect operator was already defined, it did not have a visitor implemented for any DL framework converter. Beside the implementation, we also added tests to meet the continuous integration requirements of the DNNF framework.

Concretization of inputs allows for a property to be falsified through one symbolic input such as an image, which can be changed by the falsifier, while using other concrete inputs that do not change for the purpose of testing, such as recurrent state. This functionality must be added to DNNF for each falsifier. For multi-input networks, not all inputs may be relevant to falsification. Inputs that are not used for falsification must be concretized in order to be kept constant. Anytime a back-end falsifier analyzes the network, either to determine a gradient or to validate a counterexample, it must do so using a set of inputs. By default, all of those inputs were represented symbolically, because typical networks were only concerned with one input. Code was written for the PGD falsifier to separate symbolic and concrete inputs, determine concrete values, and order them such that they were fed to the model at the correct points. The values corresponding to these inputs are defined via the property file and passed to the model alongside the symbolic input within the appropriate falsifier, which uses them for gradient attachment to the symbolic input and generation of candidate counterexamples.

Understanding, developing, and testing these two extensions to DNNF took approximately 80 hours of engineering time, resulting in 8 commits and approximately 300 lines of code, most for testing the changes.

Besides those larger changes, we carried out multiple smaller changes in DNNF. For example, supercombo's size required more memory and raising the parsing recursion limit so the parser could fit the entire model into memory at once. Another instance was the need to create more sophisticated converters between network formats to retain the required information for DNNF to work. For example, ONNX supports named inputs, but since most other frameworks rely on ordering of inputs, when converting an ONNX framework to a different one we had to support reordering of inputs based on anticipated size. Moreover, the default behavior of some existing operators had to be changed. For example, operator Conv2D had hardcoded values that assumed less complex operator behavior. Hardcoded parameter groups=1 affected the behavior of the convolutions with respect to the number of input and output channels, as well as the output shape of this operator.

4 RUNNING DNNF ON SUPERCOMBO

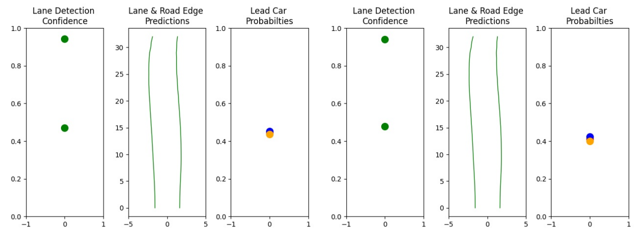
We now explore the effectiveness of our extended version of DNNF v0.1.3 at finding counterexamples for three safety properties in the supercombo model of OpenPilot version with commit hash 54d6d9.

4.1 Inputs, Outputs, and Properties

We define safety properties in terms of the inputs and outputs of supercombo[2]. The input of interest input_imgs, is a tensor comprised of two consecutive images, each 256×512 image, collected in 6-channel YUV420 format. supercombo output is a tensor of shape=(1, 6472) containing multiple estimates. Among those, we focus on the location of the lane and road edges, the confidence

[2]A full breakdown of OpenPilot models and input/output is available here: https://github.com/commaai/openpilot/tree/90af436a121164a51da9fa48d093c29f738adf6a/selfdrive/modeld/models.

(a) Image 104 **(b) Image 104 + random noise**



(c) Output for Image 104 **(d) Output for random noise**

Figure 2: Original and baseline images from the comma.ai dataset and the corresponding output from the `supercombo` model.

in those estimates, and the probabilities of lead vehicle positions. Figure 2a shows a well-formed image taken directly from the OpenPilot dataset we describe in Section 4.2 to exemplify these outputs. Three of the outputs of `supercombo` for this image are shown in Figure 2c: predicted nearest lane lines, probability for the 2 nearest lane lines, and probabilities for lead car positions identified in the image. Figure 2b shows the same image from Figure 2a with Gaussian noise applied such that the distance from the original image is ϵ=10. `supercombo` output for this image (Figure 2d) shows minimal deviation from Figure 2c. A 10-image subset of the dataset with ϵ=10 Gaussian noise applied showed similar results. This suggests random noise is not sufficient to foul `supercombo`, and that more sophisticated approaches like falsification are needed to cause significant deviations in output.

Given these features of the input and output of the network, we design three properties that can be defined informally as [3]:

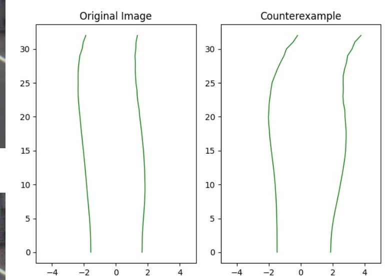
Safety Property 1: lane line confidence. – "Given an original image x and the supercombo lane confidence prediction of that image y, there is no image x' within ϵ=10 from x that renders supercombo lane confidence prediction y' such that y' is more than a 10% margin from y." A violation in this property could cause OpenPilot to confuse lane lines or disengage following a decrease in confidence.

Safety Property 2: location of nearest lane lines. – "Given an original image x and the supercombo lane prediction of that image y, there is no image x' within ϵ=10 for which the last y-value of the near left and near right lane lines in y is farther than a quarter of a standard lane width from the output of x'." This property determines the overall shape of the lane occupied by the vehicle in the upcoming timesteps. A violation could cause a lane departure.

[3]The formal properties represented in DNNP, and the code to run them, can be found in the paper repository: https://github.com/MissMeriel/openpilot-falsification.

(a) Image 199

(b) Property 2 CE



(c) Output for Property 2 CE showing lane line predictions for upcoming timesteps.

Figure 3: Original dataset image 199 and CE within ϵ=10 and the corresponding output from the `supercombo` model.

Safety Property 3: confidence of locations of lead car(s). – "Given an original image x and the supercombo lead car confidence prediction of that image being y, there is no image x' within ϵ=10 for which the lead car location confidence in y is farther than a 10% margin from the corresponding lead car location confidence in the output of x'." A violation of this property could cause a collision, a correlated shift in lead car locations, or a disengagement of OpenPilot.

4.2 Finding Counterexamples

Given the specified properties, we run DNNF v0.1.3 on `supercombo` network using input images from the Openpilot public dataset [10]. The dataset consists of over 7 hours of driving data and 522,434 images, predominantly on highways.

Figure 3 shows the results of falsification for ϵ=10 over Property 2. These images look relatively similar, in that all of the features of the original image are still discernible to the human eye[4]. Additionally, the counterexample (CE) produced for Property 2, shown in Figure 3b, seems "grainy" but is still human-interpretable.

For the Property 2 CE output shown in Figure 3c, the rightward shift in the last index of the near right lane line pushes it 1.118 meters to the right. This large delta in the CE ouput exceeds the quarter of a lane width specified in the property (0.925 meters) by 0.193 meters. The CE has also pushed the near left lane line slightly closer to the center by a much smaller amount (0.302 meters to the right). The shift in near left and right lane lines creates the appearance of a slight curve to the right in the road which is not visible in the original image.

Table 2 shows the performance of falsification for the three properties on a subset of 10 images with well-formed output, averaged over 10 runs for each image. Each run was given 100 steps to find a CE, and 100 random restarts per run. As shown in column 1, all three properties show nonzero falsification rates indicating how many of the 100 runs per property resulted in CEs. The low falsification rate for Property 1, coupled with frequent restarts, suggests

[4]Note that, because consecutive images are near-identical, we only show the first image from the two-image composite tensor input to the network. Due to space constraints, we only demonstrate property falsification on image 199 in this paper. Further results can be found in the paper repository.

	Falsification Rate	Avg. Total Time (s)	Avg. Restarts	Images with CEs
Property 1	17%	222.6	88.2	4
Property 2	47%	123.6	60.3	6
Property 3	58%	197.4	46.5	6

Table 2: Average falsification performance for ten dataset images within ϵ=10

that Property 1 is most difficult to falsify. In column 2, the average total time includes the time to reduce the network, simplify the property, and find a CE using falsifier PGD. The average falsification time for each property, respectively 142.8s, 44.6s, and 46.5s, compared to total time shows that network reduction and property simplification are by far the most expensive aspects of the falsification process. The high frontend time cost suggests avenues for future work in optimization, especially for larger networks and more complex properties. In column 3, average restarts reflects the amount of searching of the input space within ϵ=10 was needed to find a CE, suggesting the properties are ordered in descending difficulty. The confidence of nearest lane line locations appears the most difficult to manipulate, and confidence of lead car positions the easiest. This is supported by falsification rate. Lastly, column 4 shows how many of the 10 images for which a CE was found. Only one image (Image 104, shown in Figure 2a) produced CEs for all 3 properties, while 5 images produced CEs for 2 of the 3 properties. Nine of the 10 images showed a CE for at least 1 property. The lower time to run falsification on Property 2 than Property 3, despite the higher number of restarts, can be attributed to the higher number of bounds on output values for Property 3 as Property 3 bounds 3 values instead of 2. Similarly, Property 1 bounds only 2 output values, and has a comparable average total time to run despite almost double as many restarts as Property 3.

5 LESSONS LEARNED

This study provides several insights in running falsifiers on “real-world” networks. With an engineering cost on the scale of tens of hours, we were able to run the falsifier on a highly complex, real-world network and find interesting counterexamples for 3 properties that leverage aspects of the system within which the network operates. The results on falsification show that counterexamples exist within the estimated input space.

As suggested by Table 1, complex networks have unique considerations, and additional functionality must be introduced to falsifiers to support them. Adding new operators is low-cost, though operators that complicate the internal data flow of the network may require careful refactoring and testing. Managing network inputs during falsification is a larger effort, as it affects the construction of properties and the functionality of each falsifier differently. Furthermore, some large networks may exceed the recursion limit of the average machine; parsing networks in a non-recursive manner may be required to consume these networks. For almost any complex network, the problems with supporting them aren’t technical reasons; the dominant issue is engineering cost.

This project has inspired several recommendations for future work. Further investigation of large- versus small-scale network considerations might facilitate development and deployment of falsification tools. For example, the `Split` operator is present in the OpenPilot `dmonitoring` model, and is also part of the BigGAN architecture. This suggests that there may be classes of operators that are only used on large-scale networks. Other operators, such as `Slice` and `LSTM`, appear in large and/or complex networks such as AdmiralNet and need support or extended functionality in the DNNF framework. `Slice` supports weights passed as functions, requiring special parsing on the part of the DNNF frontend.

There are pending challenges regarding input space complexity as well. Our current solution for input concretization works for properties with one symbolic and multiple concrete inputs, but does not handle multiple symbolic inputs and may not generalize to multiple concrete inputs of identical size. Inputs with dependencies, like an image and a recurrent state for a state-dependent system, would need careful property construction and likely require both inputs to be symbolic, which DNNF does not currently support. Furthermore, concrete inputs have the potential to better simplify networks and reduce properties, and may ultimately reduce overhead. Constant propagation and partial input execution for multi-input networks have the potential for greater improvement of DNNF.

6 ACKNOWLEDGEMENTS

We are thankful to the developers of the DNNF toolchain.

REFERENCES

[1] Arvind Adimoolam, Thao Dang, Alexandre Donzé, James Kapinski, and Xiaoqing Jin. 2017. Classification and coverage-based falsification for embedded control systems. In *International Conference on Computer Aided Verification*. Springer, 483–503.

[2] Cas JF Cremers. 2008. The Scyther Tool: Verification, falsification, and analysis of security protocols. In *International conference on computer aided verification*. Springer, 414–418.

[3] Tommaso Dreossi, Alexandre Donzé, and Sanjit A Seshia. 2019. Compositional falsification of cyber-physical systems with machine learning components. *Journal of Automated Reasoning* 63, 4 (2019), 1031–1053.

[4] Daniel J Fremont, Johnathan Chiu, Dragos D Margineantu, Denis Osipychev, and Sanjit A Seshia. 2020. Formal analysis and redesign of a neural network-based aircraft taxiing system with VerifAI. In *International Conference on Computer Aided Verification*. Springer, 122–134.

[5] Xingwu Guo, Wenjie Wan, Zhaodi Zhang, Min Zhang, Fu Song, and Xuejun Wen. 2021. Eager Falsification for Accelerating Robustness Verification of Deep Neural Networks. In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 345–356.

[6] Greg Hogan. [n.d.]. Scaling for 10X User Growth. *comma.ai blog* ([n. d.]). https://blog.comma.ai/scaling-for-10x-user-growth/

[7] Augustus Odena, Catherine Olsson, David Andersen, and Ian Goodfellow. 2019. Tensorfuzz: Debugging neural networks with coverage-guided fuzzing. In *International Conference on Machine Learning*. PMLR, 4901–4911.

[8] ONNX. 2019. ONNX: Open Neural Network Exchange. https://onnx.ai/.

[9] OpenPilot. 2022. OpenPilot open source driver assistance system, SHA256 tag 54d6d9. https://github.com/commaai/openpilot. *OpenPilot* (2022).

[10] OpenPilot. 2022. the comma.ai driving dataset. https://github.com/commaai/research.

[11] Zi Peng, Jinqiu Yang, Tse-Hsun (Peter) Chen, and Lei Ma. 2020. *A First Look at the Integration of Machine Learning Models in Complex Autonomous Driving Systems: A Case Study on Apollo.* Association for Computing Machinery, New York, NY, USA, 1240–1250. https://doi.org/10.1145/3368089.3417063

[12] 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

[13] Xiao Wang, Saasha Nair, and Matthias Althoff. 2020. Falsification-based robust adversarial reinforcement learning. In *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 205–212.

[14] 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.* Association for Computing Machinery, New York, NY, USA, 146–157. https://doi.org/10.1145/3293882.3330579