Misbehaviour Prediction for Autonomous Driving Systems

Andrea Stocco
Università della Svizzera italiana, Lugano, Switzerland
andrea.stocco@usi.ch

Michael Weiss
Università della Svizzera italiana, Lugano, Switzerland
michael.weiss@usi.ch

Marco Calzana
Università della Svizzera italiana, Lugano, Switzerland
marco.calzana@usi.ch

Paolo Tonella
Università della Svizzera italiana, Lugano, Switzerland
paolo.tonella@usi.ch

ABSTRACT

Deep Neural Networks (DNNs) are the core component of modern autonomous driving systems. To date, it is still unrealistic that a DNN will generalize correctly to all driving conditions. Current testing techniques consist of offline solutions that identify adversarial or corner cases for improving the training phase.

In this paper, we address the problem of estimating the confidence of DNNs in response to unexpected execution contexts with the purpose of predicting potential safety-critical misbehaviours and enabling online healing of DNN-based vehicles. Our approach SelfOracle is based on a novel concept of self-assessment oracle, which monitors the DNN confidence at runtime, to predict unsupported driving scenarios in advance. SelfOracle uses autoencoder- and time series-based anomaly detection to reconstruct the driving scenarios seen by the car, and to determine the confidence boundary between normal and unsupported conditions.

In our empirical assessment, we evaluated the effectiveness of different variants of SelfOracle at predicting injected anomalous driving contexts, using DNN models and simulation environment from Udacity. Results show that, overall, SelfOracle can predict 77% misbehaviours, up to six seconds in advance, outperforming the online input validation approach of DeepRoad.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**.

KEYWORDS

misbehaviour prediction, testing, deep learning, anomaly detection

ACM Reference Format:
Andrea Stocco, Michael Weiss, Marco Calzana, and Paolo Tonella. 2020. Misbehaviour Prediction for Autonomous Driving Systems. In *42nd International Conference on Software Engineering (ICSE '20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3377811.3380353

1 INTRODUCTION

Self-driving cars are one of the emerging technologies nowadays, and possibly the standard way of transportation in the future. Such autonomous driving systems receive data from a multitude of sensors, and analyze them in real time using Deep Neural Networks (DNNs) to determine the driving parameters for the actuators.

To test such complex software systems, companies perform a limited number of expensive in-field tests, driving a car on real world streets, or within closed-course testing facilities [13]. This provides detailed sensor data of the vehicle that are recorded, played back, and recreated within a simulator to obtain comprehensive test scenarios. Simulation-based test scenarios allow re-testing new autopilot releases on a large numbers of nominal conditions, as well as challenging (e.g., adverse weather) and dangerous circumstances (e.g., a pedestrian suddenly crossing the road), at a low cost [13].

The potentially unlimited number of testable driving scenarios, combined with the lack of human interpretability of the internal functioning of DNNs [2], makes it difficult to predict the vehicle's (mis)behaviour with respect to unforeseen edge-case scenarios. Misbehaviours span a wide range of situations, associated with different degrees of severity, from cases where the car does not drive smoothly (e.g., excessively high derivative of the steering angle over time), up to safety-critical failures and casualties [17, 42, 43]. In this respect, promptly detecting unexpected, untested execution contexts is of paramount importance, so as to make sure that human-driven or self-healing corrective actions take place to ensure safety.

In this paper, we tackle the *self-assessment oracle problem* for autonomous driving system, i.e., the problem of monitoring the confidence level of a DNN-based autonomous driving system in order to timely predict the occurrence of future misbehaviours. The problem is critical because a failure in detecting an unexpected condition may have severe consequences (i.e., a fatal crash), whereas false alarms, even if not dangerous, may cause driver's discomfort and negatively affect the driving experience. Creating a self-assessment oracle that evaluates the confidence of a DNN *at runtime*, and predicts whether the system is within a low-confidence zone, is a largely unexplored research problem.

Challenges arise because unexpected driving conditions are, by definition, unknown at training time, otherwise they would be used to train a better DNN [12]. As a consequence, the problem being addressed belongs to the unsupervised class of data analysis, and we have to infer the unexpected only by looking at the normal driving scenarios. Moreover, the ensemble of possible misbehaviours for a DNN-based system is vast and necessarily domain-dependent, being associated with deviations from the functional requirements.

In recent years, researchers have proposed solutions for testing autonomous driving systems software [1, 6, 7, 18, 19, 19, 34, 44, 55]. A number of approaches propose input generation techniques that produce adversarial or corner cases, used to improve the robustness of self-driving car modules by re-training [26, 34, 44, 55]. Other works target test case generation to expose faults for extreme conditions, such as the vehicle colliding with a pedestrian [7], or driving off the road [18]. All these approaches concern *offline* solutions for improving the robustness and reliability of DNNs, achieved by enhancing the training data and the autopilot module, which are extended to include underrepresented critical scenarios.

In this paper, we propose a novel self-assessment oracle for autonomous vehicles based on *confidence estimation*, *probability distribution fitting*, and *time series analysis*. Our technique is implemented in a tool called SelfOracle, which leverages reconstruction-based techniques from the deep learning (DL) field to analyze spatiotemporal historical driving information. The reconstruction error is used as a black-box confidence estimation for the DNN. During probability distribution fitting, SelfOracle captures the behaviour of the self-driving car under nominal conditions, and fits a Gamma distribution to the observed data. Analytical knowledge of Gamma's parameters allows SelfOracle to estimate an optimal confidence threshold, as a trade-off between prediction of all misbehaviours, and false alarms. By observing a decreasing confidence trend over time, SelfOracle can anticipate a misbehaviour by recognizing unexpected conditions timely enough to enable healing actions such as manual or automated disengagement.

We have evaluated the effectiveness of SelfOracle on the Udacity simulator for self-driving cars [47], using DNNs available from the literature. We have modified the simulator to being able to inject unexpected driving conditions (i.e., day/night cycle, rain, fog, or snow) in a controllable way. In our experiments on 72 simulations, SelfOracle is able to safely anticipate 77% out-of-bound episodes/crashes, up to 6 seconds in advance. A comparative experiment with the online input validation strategy of DeepRoad [55] shows that SelfOracle achieves substantially superior misbehaviour prediction on all the considered effectiveness metrics.

Our paper makes the following contributions:

Technique An unsupervised technique for misbehaviour prediction based on confidence estimation, probability distribution fitting and time series analysis, implemented in the tool SelfOracle, which is available [45].

Simulator An extension of the Udacity simulator to inject unexpected driving conditions dynamically during the simulation.

Evaluation An empirical study showing that the reconstruction error used by SelfOracle for time series analysis is a promising confidence metric for misbehaviour prediction, outperforming the online input validation approach of DeepRoad.

Dataset A dataset of 765 labeled simulation-based collision and out-of-bound episodes that can be used to evaluate the performance of prediction systems for autonomous driving cars.

2 BACKGROUND

DNN-based Autonomous Vehicles. Self-driving cars (SDC, hereafter) have benefited from many technological advancements both in hardware and in software. Data gathered by LIDAR sensors, cameras, and GPS are analyzed in real time by advanced DNNs which govern over the actual maneuvers of the car (i.e., steering, braking, acceleration). In order to manage a wide variety of driving scenarios, SDCs necessitate a large amount of driving data, combining nominal and adversarial scenarios [8].

To date, it is still unlikely for a DNN to generalize correctly to the plethora of driving situations met everyday by human drivers. As such, a component monitoring the *confidence* of the DNN may promptly detect when the SDC is entering a low-confidence zone, and activate healing strategies that bring the vehicle to a safe state.

In self-driving cars, depending on the level of autonomy, the self-healing procedure can either involve the human driver, or can be delegated to an automated system.[1] At both levels, early and accurate misbehaviour prediction is an essential precondition to enable safe healing, and an overall pleasant driving experience.

Confidence Measures in DNNs. The prediction must consider DNN uncertainties originating from the measurements in response to possible adverse environmental conditions in which the SDC operates. The confidence level of a SDC can be measured through white-box or black-box techniques. White-box metrics monitor the internal behaviour of a DNN component. For simple classifiers, measuring softmax probabilities, or information theoretic metrics such as entropy, and mutual information [37] may suffice. For more complex networks such as those than operate on a SDC, softmax probabilities and entropy are known to be unreliable confidence estimators [52]. Moreover, white-box metrics require a transparent access to the network, and substantial domain-knowledge for the creation of nontrivial probabilistic models that approximate the network's uncertainty.

Black-box techniques, differently, model the SDC uncertainty by monitoring the relation between the current input (images) and the input data used during training. For instance, consider a SDC which has been trained only with images representing highways. If images representing a narrow city street are given to the DNN, the model will still output steering angles, but ideally we would like to warn the SDC of a drop in the confidence level. The main advantages of black-box confidence metrics consist in being independent from the specific SDC architecture, requiring no modifications to the existing DNN model because they use information which is readily available for analysis, and in being, therefore, highly generalizable. In this paper, we focus on black-box confidence estimation. We next describe autoencoders and time series analysis, which are the main building blocks of our approach.

Autoencoders. An autoencoder (AE) is a DNN designed to reconstruct its own input. It consists of two sequentially connected components (an encoder and a decoder) that are arranged symmetrically. The simplest form of autoencoder (SAE) is a three-layer DNN: the input layer, the hidden layer, and the output layer. The hidden layer encodes any given input $\boldsymbol{x} \in \mathcal{R}^D$ to its internal representation (*code*) $\boldsymbol{z} \in \mathcal{R}^Z$ with a function $f(\boldsymbol{x}) = \boldsymbol{z}$. Usually $Z \ll D$, where Z is the dimension of encoded representation and D is the dimension of the input. The output layer (decoder) decodes the encoded input with a reconstruction function $g(z) = \boldsymbol{x'}$, where $\boldsymbol{x'}$ is the reconstructed input $\boldsymbol{x}$. The autoencoder minimises a loss

[1]https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety

function $\mathcal{L}(\boldsymbol{x}, g(f(\boldsymbol{x})))$, which measures the distance between the original data and its low-dimensional reconstruction. A widely used loss function in autoencoders is the Mean Squared Error (MSE).

The input and output layers of autoencoders have the same number of nodes. If multiple hidden layers are used, the architecture is referred to as deep autoencoder (DAE). The surge of novel kinds of DNNs has correspondingly produced variants of autoencoders based on such architectures. For example, convolutional autoencoders [28] allow learning powerful spatial-preserving relationship within images at a lower training time with respect to fully dense layers. Another interesting proposal are variational autoencoders (VAE) that are able to model the relationship between the latent variable $\boldsymbol{z}$ and the input variable $\boldsymbol{x}$ by learning the underlying probability distribution of observations using variational inference [3].

Time Series Analysis. Traditional feedforward DNNs assume that all inputs and outputs are independent of each other. However, learning temporal dependencies between inputs or outputs is important in tasks involving continuous streams of data. Thus, a predictive model can take advantage of information from the previous inputs/outputs, to enhance its predictive capability.

Time series analysis can be applied to the output sequence produced by a DNN to identify the trend and predict future values. Among the numerous models available for time series analysis, the most widely used ones are autoregressive (AR), integrated (I) and moving average (MA) models, along with their combinations. An AR model of order k predicts the next value x_t as a linear combination of past values $x_{t-1}, \dots, x_{t-k}$:

$$x_t = \alpha_0 + \sum_{i=1}^{k} \alpha_i x_{t-i} + \epsilon_t \tag{1}$$

where coefficients $\alpha_0, \dots, \alpha_k$ can be estimated by the least square method and ϵ_t represents the error term.

Processing of a sequence of inputs can be achieved by recurrent neural networks (RNN) equipped with long short-term memory (LSTM) [22], which is capable of dealing with both short and long range dependencies. In LSTM, outputs are influenced not only by the current input but also by the state of the RNN, which encodes the entire history of past inputs.

3 PROBLEM FORMULATION

We focus on SDCs that perform *behavioural cloning*, i.e., the DNN learns the *lane keeping* [18] behaviour from a human driver. Models such as the ones by NVIDIA [9] or the Udacity self-driving challenge [48] are trained with visual inputs (i.e., images) from car-mounted cameras that record the driving scene, paired with the steering angles from the human driver. The DNN then "learns how to drive" by discovering underlying patterns within the training images representing the shape of the road, and predicting the corresponding steering angle.

For classification problems (e.g., hand-written digit recognition), misbehaviours can be defined as inputs that can be confidently labeled by humans while they are misclassified by a DNN. Differently, for regression problems such a definition is more challenging, because there is no expected outcome for an individual output of the DNN, and it is only the overall behaviour resulting from the DNN predictions that may or may not be acceptable, depending on the specific application domain.

In steering angle prediction, it is challenging to decide if the steering angle produced by a DNN is wrong, because the optimal steering angle is generally unknown for a new test scenario and even if it were known, the amount of difference between predicted and expected steering angle that qualifies as an error is difficult to decide a priori. It is instead more realistic to figure out whether a *chain of inaccurate predictions* ultimately leads to a misbehaviour, because of the cumulative prediction errors.

In fact, the very definition of misbehaviour should be decoupled from the notion of correct/wrong DNN output, being instead linked to the ability of a DNN of abstracting from the training examples and learning how to drive in different ways/conditions. It is the task of the DNN to generalize the training knowledge to make the model robust with respect to slightly different conditions from those observed in the training set, for example adapting the driving style to different weather conditions.

3.1 Misbehaviour of Autonomous Vehicles

In the context of this paper, a DNN exhibits a *misbehaviour* in a given test scenario if the overall system that contains the DNN does not respect its requirements due to the outputs produced by the DNN. In the autonomous driving domain, there are many possible misbehaviours, associated with the different requirements that such systems are supposed to realize. Safety requirement violations are by far the most critical requirements, as a misbehaviour in the steering component may cause a crash of the vehicle with potential casualties. However, in general, a SDC might violate also other driving requirements, e.g., related to ride comfort [11], such as excessive derivative of the steering angle, unstable movement around the centerline, or excessive deceleration.

In this paper, we focus on the prediction of two safety-critical misbehaviours: (1) collisions and (2) out-of-bound episodes (OBEs). The rationale for this choice are as follows. First, they represent the vital requirement to be satisfied and thoroughly tested (i.e., the car should stay in lane and avoid whatsoever collision), without which autonomous driving vehicles would be hardly accepted in production. Second, leveraging a simulation environment such as Udacity's [47] allows us to: (1) safely test such critical scenarios, and (2) precisely define, observe and measure them, in order to support crash analysis and reproduction.

To conclude, the problem we aim to address in this paper is predicting when a self-driving car is within a low-confidence area because of an unexpected execution context, timely enough to take countermeasures before the vehicle crashes or drives off road.

4 APPROACH

The goal of our approach is to monitor the confidence level of a SDC as it runs, and to promptly predict whether drops in confidence correlate with potential future misbehaviours. Our approach works in an end-to-end fashion, analyzing directly the input data as retrieved by the car (an image from the center camera), making the approach independent from the specific architecture of the self-driving component, requiring no modifications to the existing DNN model, and being therefore, highly generalizable.

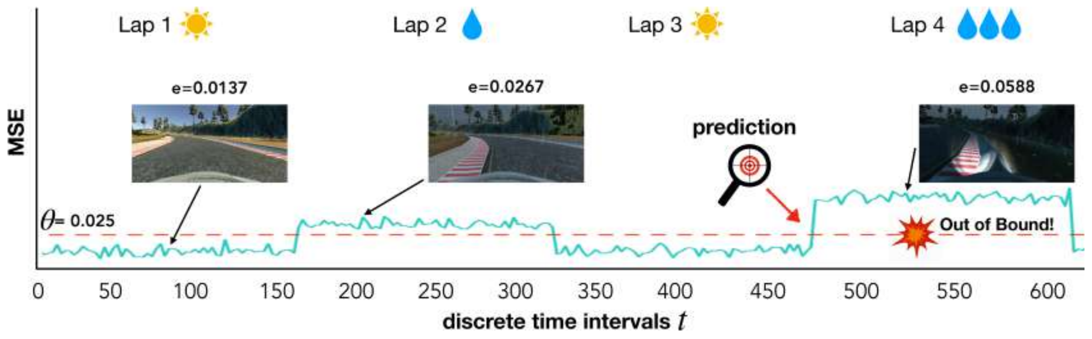

Figure 1: Confidence levels of Nvidia's DAVE-2 [9] in response to changing driving scenarios. The picture shows frames captured during the execution of the SDC along one of our testing tracks, under different conditions: (Lap 1) sun, (Lap 2) light rain, (Lap 3) sun, and (Lap 4) heavy rain. The picture juxtaposes the reconstruction error by the anomaly detector of SelfOracle, which is used as a proxy for the DNN confidence. We can notice that the reconstruction error is low when the car drives under sunny conditions (i.e., conditions similar to those observed during training), whereas the error increases moderately with adverse conditions (the car does no longer follow the center of the road), and grows above a given threshold when facing heavy rainy conditions at night time (which cause the SDC to drive off the road).

The main working assumption is that a prediction model trained on normal data should learn the normal time series patterns. When the model is used on a SDC in the field, it should worsen its performance as the car approaches previously unseen regions as compared to normal, known regions (Figure 1). Then, using a decision boundary mechanism we can timely alert the human driver (NHTSA Level 4) or the main self-driving component (NHTSA Level 5).

We now detail each step of our approach, which consists of two main phases: (1) *model training under nominal driving behaviour*, and (2) *field usage of the trained model*.

4.1 Training of SelfOracle under Nominal Driving Behaviour

Figure 2 illustrates the training phase of our approach, which consists of several steps.

4.1.1 Reconstructor. The first step consists in retrieving a model of normality from the training driving scenarios. Thus, in the training set, we capture the visual input stream of the SDC under nominal situations. Then, we train our driving scenario reconstructor with such "normal" instances. The motivation for the use of reconstructors is due to the results obtained in the field of anomaly detection by reconstruction-based techniques. Particularly, autoencoders have proven effective as anomaly detectors because the proximity in their latent space translates into low reconstruction error [3, 31, 36, 54], and are computationally very efficient.

Let us consider a training set $\boldsymbol{X} = \{x_1, x_2, \ldots, x_n\}$ of n image frames, where the index $i \in [1 : n]$ of $x_i \in \boldsymbol{X}$ represents the discrete time t. Depending on the considered architecture, a reconstructor can be *singled-image* or *sequence-based*. For singled-image reconstructors, only one image frame is considered at a time. When the discrete time is $t = i$, x_i is the input and the reconstructor recreates it into x'_i. For sequence-based reconstructors, assuming k image frames preceding x_i are used to reconstruct x_i, the sequence $\langle x_{i-k}, \ldots, x_{i-1} \rangle$ is the input used to output x'_i, a prediction of the actual current frame x_i. For instance, for $k = 3$ and $i = 4$, the reconstructor considers the sequence $\langle x_1, x_2, x_3 \rangle$ in order to predict the current frame x_4.

At the end of this task, each reconstruction error $e_i = d(x_i, x'_i)$ can be computed, where d is a proper distance function, such as the Euclidean distance. This results in the set of reconstruction errors $\boldsymbol{E} = \{e_1, e_2, \ldots, e_n\}$, available for all elements in the training set $\boldsymbol{X}$.

4.1.2 Probability Distribution Fitting. We build a model of normality for the reconstruction errors $\boldsymbol{E} = \{e_1, e_2, \ldots, e_n\}$ collected in nominal driving conditions and automatically determine a threshold θ that brings the expected false alarm rate in nominal conditions below some acceptable, configurable level ϵ (e.g., $\epsilon = 10^{-3}$ or $\epsilon = 10^{-4}$). To this end, we use probability distribution fitting to obtain a statistical model of normality. Indeed, using the raw reconstruction error distribution is unreliable, because high error values are rare as such distribution is obtained from nominal data. Therefore, any estimation above a reasonably high threshold becomes inaccurate if done directly on the raw data, because the estimated false alarm rate would be incorrectly assumed to be equal to zero when only a few, or even no data points, are observed on the right of the chosen threshold.

The reconstruction error $e = d(x, x')$ can be computed by comparing the individual pixels of the images x and x' and taking the mean pixel-wise squared error. Assuming images have width W, height H and C channels (usually, RGB channels for colour images), the reconstruction error is defined as follows:

$$d(x, x') = \frac{1}{WHC} \sum_{i=1, j=1, c=1}^{W,H,C} (x[c][i, j] - x'[c][i, j])^2 \qquad (2)$$

We assume that the pixel-wise error $e[c][i, j] = x[c][i, j] - x'[c][i, j]$ follows a normal distribution with pixel-dependent variance: $e[c][i, j] \sim \mathcal{N}(0, \sigma_{c,i,j})$. Correspondingly, the sum of the squares of pixel-wise errors $e[c][i, j]$ follows a Gamma distribution: $e = d(x, x') \sim \Gamma(\alpha, \beta)$. We get a Gamma distribution instead of a χ^2 distribution because pixel-wise errors have different channel/pixel dependent non-unitary variances.

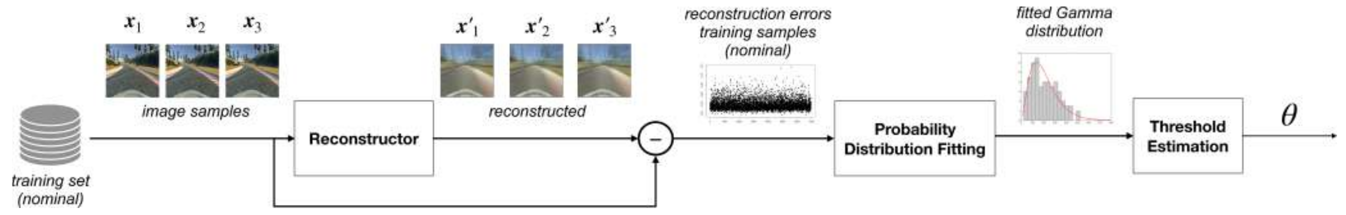

Figure 2: Model Training under nominal driving behaviour.

Definition of Gamma Distribution. Gamma is a probability model for a continuous variable on $[0, \infty)$ which is widely used in engineering, science, and business, to model continuous variables that are always positive and have skewed distributions [23]. The *probability density function* of a random variable $x \sim \Gamma(\alpha, \beta)$ is:

$$f(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x} \qquad x > 0;\ \alpha, \beta > 0 \tag{3}$$

where α is the *shape* parameter (which affects the shape of the distribution), β is the *rate* parameter (or inverse scale, which stretches/shrinks the distribution) and Γ is the Gamma function. When α is large, the Gamma distribution closely approximates a normal distribution with the advantage that the Gamma distribution has non-zero density only for positive real numbers.

The *Gamma function* Γ can be seen as a solution to the interpolation problem of finding a smooth curve that connects the points (n, m) with $m = (n-1)!$ at any positive integer value for n. Such a definition was extended to all complex numbers with a positive real part by Bernoulli, as a solution to the following integral:

$$\Gamma(z) = \int_0^\infty x^{z-1} e^{-x} dx \qquad \Re(z) > 0; \tag{4}$$

Fitting the Gamma Distribution. One effective method to estimate the parameters of a distribution that best fit the data is by maximum likelihood estimation (MLE) [14], which we briefly report next. The likelihood function reverses the roles of the variables: in Equation 3, the values of x are known, and are the fixed constants, whereas the unknown variables are the parameters α and β. MLE involves calculating the values of these parameters so as to obtain the highest likelihood of observing the values of x when the given parameters are supplied to f.

Under the assumption of independence of the data, the likelihood of the data given the parameters of the distribution is conveniently defined as the logarithm of the joint probability of the data for a given choice of the parameters. In the case of a Gamma distribution, with a dataset consisting of reconstruction errors $\boldsymbol{E} = \{e_1, e_2, \ldots, e_n\}$, we get:

$$\mathcal{L}(\alpha, \beta; \boldsymbol{E}) = \frac{1}{n} \sum_{i=1}^{n} \log f(e_i | \alpha, \beta) = \tag{5}$$

$$n(\alpha - 1)\overline{\log e} - n \log \Gamma(\alpha) - n\alpha \log \beta - n\overline{e}/\beta \tag{6}$$

where $\overline{e}$ ($\overline{\log e}$) is the mean (log) reconstruction error over $\boldsymbol{E}$. To find the values of parameters α and β that maximize $\mathcal{L}$ we have to find a solution to the equations: (1) $\partial\mathcal{L}/\partial\alpha = 0$; (2) $\partial\mathcal{L}/\partial\beta = 0$. The second equation can be easily solved analytically, resulting in $\beta = \overline{e}/\alpha$. By substituting the value of β into the first equation, we get an equation that unfortunately cannot be solved analytically. However, the Newton method can iteratively converge to the solution quite quickly. The output of such numerical estimation will be the pair of parameters α and β of the Gamma distribution that best fit the reconstruction errors.

Example of Threshold Estimation. Let us consider a set of reconstruction errors. Figure 3 shows the histogram of those produced on the DAVE-2 dataset when the reconstructor is a VAE. On such dataset, the MLE method estimates the following Gamma parameters: $\alpha = 15$; $\beta = 392$. Figure 3 shows the Gamma distribution obtained with such parameter values. Let us to set a false alarm rate $\epsilon = 10^{-2}$. The threshold θ with a probability mass above the threshold equal to 10^{-2} can be easily obtained as the inverse of the cumulative Gamma distribution $F(x)$: $\theta = F^{-1}(1 - \epsilon)$. This ensures that the cumulative probability of values $\leq \theta$ is $1 - \epsilon$, leaving only a probability of ϵ to the tail following θ. We use the estimated θ as threshold to distinguish anomalous conditions (reconstruction error $\geq \theta$) from normal ones (reconstruction error $< \theta$).

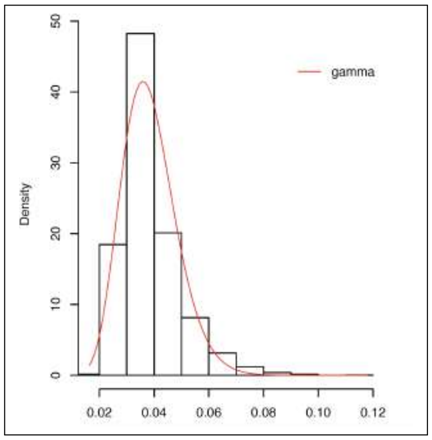

Figure 3: Fitted Gamma distribution of reconstruction errors from a VAE on the DAVE-2 dataset.

4.2 Usage Scenario

Figure 4 shows how SelfOracle is used online for misbehaviour prediction after model training (i.e., after fitting the Gamma distribution and estimating the threshold θ). Misbehaviour prediction is executed online as the SDC drives. In this phase, the SDC generates data continuously and the reconstructor recreates the incoming stream of images. The sequence of reconstruction errors is passed

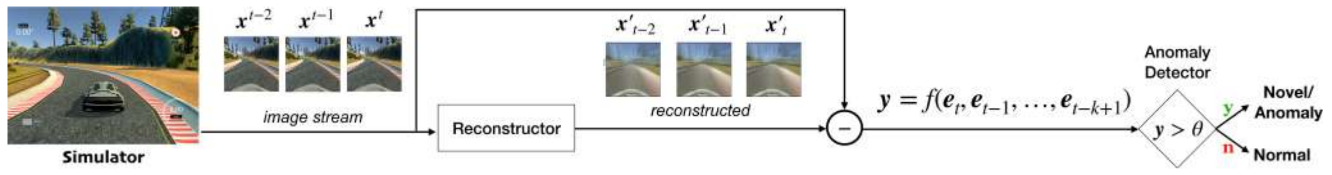

Figure 4: Usage Scenario of SelfOracle.

through the autoregressive model f and the resulting, filtered error is compared against the threshold θ, which determines whether an anomaly is detected or not. In the former case, self-healing is triggered and the SDC is brought to a safe state.

4.2.1 Time-aware Anomaly Score Prediction. The reconstruction error e_t at time t might be susceptible to single-frame outliers, which are not expected to have a big impact on the driving of the car, but would indeed make the misbehaviour predictor falsely report an anomalous context. For this reason, we smooth such noisy oscillations by applying an autoregressive filter: the sequence of reconstruction errors is passed to a module that performs time series analysis. In Figure 4, this corresponds to the AR filter f. The output of this filter, instead of the raw reconstruction error e_t, is compared with the threshold θ to recognize unexpected driving conditions. In our experiments we used a simple AR model (see Equation 1) with $\alpha_0 = 0$ and $\alpha_i = 1/k$ for $i = 1, \ldots, k$, i.e., the arithmetic mean of reconstruction errors over the last k frames.

5 EMPIRICAL EVALUATION

5.1 Research Questions

We consider the following research questions:

RQ$_1$ (effectiveness): How effective is SelfOracle in predicting anomalies for autonomous vehicles? What are the best reconstructors to use?

RQ$_2$ (prediction): How does the misbehaviour predictions of SelfOracle change as we increase the reaction period (i.e., we anticipate the time of prediction)?

RQ$_3$ (comparison): How does SelfOracle compare with DeepRoad's [55] online input validation?

5.2 Self Driving Car Models

We evaluate our framework on three existing DNN-based SDCs: Nvidia's DAVE-2 [9], Epoch [41], and Chauffeur [40]. We choose these models because they are robust SDC models and they are publicly available, thus they can be trained and evaluated on the simulator. Moreover, they have been objects of study of other testing works [34, 44]. DAVE-2 consists of three convolutional layers, followed by five fully-connected layers. Chauffeur uses a CNN to extract the features of input images, and a RNN to predict the steering angle from 100 previous consecutive extracted features. Epoch is implemented as a simple CNN with three convolutional layers.

5.3 Simulation Platform

As common industrial practices require [13], for our empirical study we used a simulation environment of the whole system in operation. The motivation for this choice is twofold. First, unlike previous works [34, 44, 55], we cannot rely on existing driving image datasets such as the ones released by Udacity [49], because they lack any episode of crash, or cars driving off road whatsoever. Indeed, a major problem in anomaly detection research is the lack of labeled benchmark datasets [12], and the self-driving car domain is no exception. Second, for online testing, our definition of misbehaviour (Section 3) requires the creation of a set of "controllable" unexpected conditions that may potentially cause them, along with a way to precisely record them. The use of a simulation platform is viable according to a recent paper by Haq et al. [20], showing that simulator-generated data yield similar prediction errors as those obtained on real-world datasets, and that offline testing is less viable in exposing safety violations that occur during in-field testing. Thus, we investigated the effectiveness of our approach in predicting safety-critical misbehaviours in the Udacity simulator [47]. The Udacity simulator is developed with Unity [51], a popular cross-platform game engine. The simulator provides two default tracks for testing DNNs models. The simulator executes in two modes: (1) *training mode*, in which the user manually controls the car while the simulator records her actions, and (2) *autonomous mode*, in which the car is controlled by an external agent, such as a DNN-based autonomous driving system. Moreover, we added a third track [46] to the existing set, and we implemented two additional components, namely, an *unexpected context generator*, and a *collision/OBE detection system.*

5.3.1 Unexpected Context Generator. First, we developed a method to gradually inject *unseen* conditions during autonomous mode (i.e., conditions diverse from the training mode's defaults). The first condition deals with illumination. Our extension of the simulator can gradually change the lightning condition of the track, simulating the passage between day and night (*day/night cycle*). The component that controls illumination can produce changes in a gradual way, is applicable to all tracks, and is customizable in order to increase or decrease the brightness (or darkness), as well as control the duration of a simulated day/night cycle. The sun and the moon objects are rotated around the zero point vector according to the current time of the day by extracting how many degrees the celestial bodies should rotate after each update interval. This effect produces realistic changes in the shadows of *all* objects in the scene, night sky, and illumination brightening and dimming. We fixed the retro illumination when the sun is positioned below the scene by deactivating it when the Y axis component becomes negative. The sun is switched off at sunset and switched on at sunrise accordingly, and intensities during the simulations are varied by applying linear interpolation from the minimum to the maximum intensity along the chosen day length (60 s in our experiments).

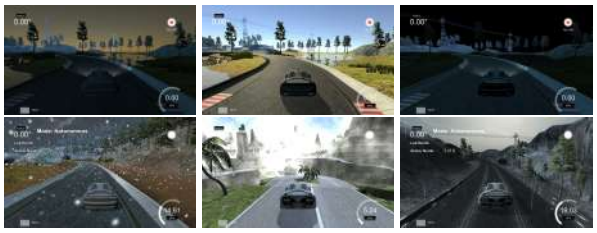

Figure 5: (top) Day/night cycle (sunrise, day, night) and (bottom) weather effects (snowy Lake Track, foggy Jungle Track, and rainy Mountain Track).

The second kind of unseen environmental condition relates to *weather effects*. We implemented rain, snow and fog effects with a variable intensity during the simulation. For the implementation, we used a specific Unity component called Particle System [32] which can simulate the physics of a cluster of particles with high performance. The particle system spawns particles according to a predefined and optimized algorithm (such as per point, area, volume), and the particles can be updated only through fixed values according to fixed events (update particles color, size, direction, speed, or acceleration when a timer ends, a collision occurs, or randomly). After creating the rain, snow and mist effects, we wrote a script to localize the particles around the car in motion, handling the movement of the effects in response to that of the car, and to control the intensity of the effect. It applies linear interpolation to decrease or increase the number of particles produced over time, visually changing the impact of the effect. Specifically, the rain particles emission rate ranges between a minimum of 100 (light rain) to a maximum of 10,000 particles/s (heavy rain); fog between [100;2,000] particles/s, and snow between [100;800] particles/s. The simulation platform is also in charge of changing the sun intensity, the sun color and the sky box according to the selected effect. Figure 5 shows a few examples.

5.3.2 Collision/OBE Detection System. Following our definition of safety-critical misbehaviours, we implemented an automated collision/OBE detection system (ACODS) that records any unwanted interaction of the SDC with the environment, allowing us to experiment the effectiveness of SelfOracle at anticipating such episodes during the occurrence of unexpected scenarios (Section 5.3.1).

We implemented ACODS based on *colliders*, which are consolidated building blocks of modern game engines to simulate the physical interaction between objects. We approximate the car body with a geometry mesh, and implemented a collider callback that informs the simulator of any physical interaction of the car with scene objects. When the car "hits" the road, it means the car is actually on track. Differently, when the car "collides" with any other object, the callback registers whether it is a crash against some object (Figure 6 (left)), or whether it is an OBE (Figure 6 (right)).

We also implemented an automatic restart mechanism that restores the SDC to a safe position after a crash/OBE, allowing us to record multiple simulations without the need for manual restart. This was implemented leveraging track waypoints, i.e., phantom objects that are used to mark certain positions and directions in a 3D scene. Essentially, each consecutive pair of waypoints w_n, w_{n+1} defines a track sector (which also gives us a way to monitor and control the length of the simulations). Automated restart is performed as follows. If the car crashes at some sector w_n, w_{n+1}, we tag the sector as not complete, and the system automatically moves the car to the waypoint w_{n+1}, adjusting position and angle according to the waypoint orientation.

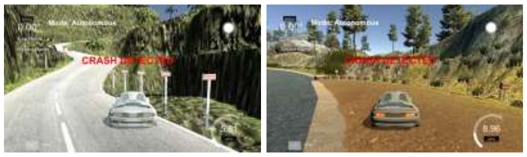

Figure 6: Simulator crash/OBE detection.

5.4 Procedure

5.4.1 Data Generation (Training Set). Training data were collected by the authors in training mode by performing 10 laps on each track, following two different track orientations (normal, reverse). Overall, we obtained a dataset of 124,638 training images (at 10-13 fps), divided as follows: 32,243 for Track 1 (Lake), 51,422 for Track 2 (Jungle), and 40,973 for Track 3 (Mountain). Differences depend on the track lengths. To allow a smooth driving and a correct behaviour capture (i.e., lane keeping), the maximum driving speed was set to 30 mph, the default in the Udacity simulator.

5.4.2 SDC Model Setup & Training. All SDCs models were trained on 124,638 images from the three cameras. We used data augmentation as a consolidated practice for building more reliable and generalizable SDCs, limiting the lack of image diversity in the training data. Specifically, 60% of the data was augmented through different image transformation techniques (e.g., flipping, translation, shadowing, brightness). We cropped the images to 66 x 200, and converted them from RGB to YUV colour space. All SDC models were trained for 500 epochs with batch size of 256 on a machine featuring an i9 processor, 32 GB of memory, and an Nvidia GPU GeForce RTX 2080 Ti with 11GB of memory. Basically, the training was meant to create solid models for testing, i.e., able to drive multiple laps on each track under nominal conditions without showing any misbehaviour in terms of crash/OBE.

5.4.3 Evaluation Set. To collect the evaluation data, we executed 72 simulations (3 SDC x 8 conditions x 3 tracks in autonomous mode), each consisting of two laps. As in the data generation phase, the maximum speed was set to 30 mph. Specifically, for each SDC and for each track, we performed one simulation in the same normal conditions as the training set. This allowed us to estimate the number of false alarms (false positives) in nominal conditions. Second, we performed four simulations activating in turn a single unexpected condition: day/night cycle, rain, snow, fog. Third, we performed three simulations activating in turn a combined condition: day/night cycle + rain, day/night cycle + snow, day/night cycle + fog.[2]

[2]All such conditions may actually occur in nominal runs as well. There is no special reason for this specific choice of normal/anomalous conditions and different permutations would be of course allowed (e.g., normal = day/night + rainy; anomalous = snowy). The only important prerequisite is that the chosen anomalous condition is unseen at training time.

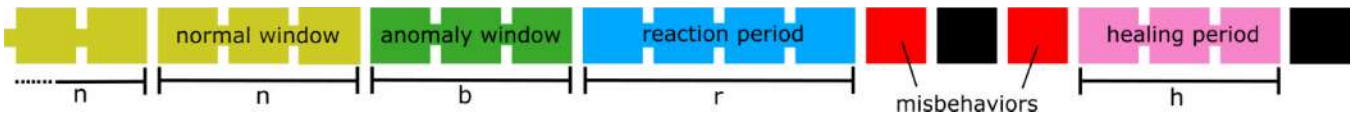

Figure 7: Labelling of anomalous and normal windows in driving image stream.

In our experiments, we used a value of 60 s both for the day/night cycle and for the loop between the minimum and maximum intensity of the effects. This value was chosen empirically given the relatively short speed of the SDC, and the small length of the tracks, allowing us to test the behaviour of the SDC on each of the tracks' subsets under all possible conditions. For example, the bridge part of Track1 (Lake) has been driven on under both dawn/day/-sunset/night conditions (day/night cycle) and minimal/maximal intensity of rain/fog/snow.

Overall, we obtained a dataset of 778,592 images, divided as follows: 188,032 for Track 1 (Lake), 260,064 for Track 2 (Jungle), and 208,064 for Track 3 (Mountain) with unknown conditions and 33,088 for Track 1, 46,272 for Track 2 and 43,072 for Track 3 in known conditions.

5.4.4 SelfOracle's Configurations. We used four autoencoders taken from existing guidelines [24]: (1) *SAE* (simple autoencoder with a single hidden layer), (2) *DAE* (deep five layers fully-connected autoencoder), (3) *CAE* (convolutional autoencoder alternating convolutional and max-pooling layers), and (4) *VAE (variational autoencoder).* All autoencoders take as input a single image. We added images taken by the side cameras of the car (left, right) to allow better generalization. Lastly, we performed further data augmentation on 60% of the inputs, as described in Section 5.4.2.

As an additional sequence-based reconstructor, we also implemented an LSTM consisting of two LSTM-layers and one convolutional layer. All our code is publicly available in the replication package accompanying this paper [45].

5.4.5 Baseline. We use the input validation technique of DeepRoad [55] as baseline for SelfOracle. Due to the unavailability of an open-source version of such technique, we implemented of our own version based on the authors' description, which is publicly available in our replication package [45]. For input validation, DeepRoad uses the pre-trained VGG19 ImageNet classifier [38] to extract style and feature vectors from a given image. Principal Component Analysis (PCA) is then used to reduce all style and feature vectors, concatenated into a matrix, to three dimensional representations, which support distance/similarity estimation. To allow a fair comparison, we integrated it within SelfOracle as reconstructor. However, unlike autoencoders, DeepRoad is computationally very expensive and memory demanding (due to the size of the matrix supplied to PCA). In the paper, authors reduced their training set to 600 images, which were resized to 120x90. During input validation, the three dimensional representation of an online input image is compared to the nominal images by measuring the average of the top-100 minimum distances from the training set.

Our own implementation relaxed the restrictions above by considering a training set consisting of 3,000 randomly sampled images (i.e., 5× improvement with respect to the original implementation described in the paper), resized to 224x224, which is the default input size for VGG19 [25]. Keeping the fraction of the training set constant ($\frac{1}{6}$), we compute similarity based on the average of the top-500 minimal distances.

5.5 Evaluation of Simulation Results

We evaluated all approaches offline, by splitting the evaluation set of recorded images in *windows of consecutive frames*, which we labelled as either anomalous or normal (Figure 7). In anomalous windows, SelfOracle is expected to predict the shortly-following misbehavior.

5.5.1 Labelling of Anomaly and Normal Windows in Evaluation Data. Let $X = \{x_1, x_2, \dots, x_n\}$ be the sequence of considered images (frames). Misbehaviors are represented as $m_j \in \{0, 1\}$, where $m_j = 1$ iff a misbehavior is recorded at $x_j \in X$. We define a healing period as a sequence of h misbehaviour-free frames following a misbehaviour at time t. We define a reaction period as a sequence of r misbehaviour-free frames preceding a misbehaviour at time t' and not intersecting any healing period. We define an *anomalous window* as *a* consecutive misbehaviour-free frames followed by a reaction period that does not intersect any healing period. We define a *normal window* as b consecutive misbehaviour-free frames followed by an anomalous window, or a normal window that does not intersect any healing period. This is illustrated graphically in Figure 7. Formally, assuming any misbehavior recorded at time $t \in [1{:}n]$, i.e., $m_t = 1$:

- window x_{t+1} to x_{t+h} is labelled as healing period if $m_t = 1$ and $m_j = 0\ \forall j \in [t+1{:}t+h]$;
- furthermore, the window x_{t+1} to x_{k-1} is also labelled as healing period if $m_k = 1$ with $k > t$ and $k - t - 1 < h$, and $m_j = 0\ \forall j \in [t+1{:}k-1]$;
- window x_{t-r} to x_{t-1} is labelled as reaction period iff $m_t = 1$, $m_j = 0\ \forall j \in [t-r{:}t-1]$ and no healing period contains any frame from x_{t-r} to x_{t-1};
- window x_i to x_{i+a-1} is labelled as anomaly window iff a reaction period starts at x_{i+a}, $m_j = 0\ \forall j \in [i{:}i+a-1]$ and no healing period contains any frame from x_i to x_{i+a-1};
- window x_i to x_{i+b-1} is labelled as normal window iff an anomaly or a normal window starts at x_{i+b}, $m_j = 0\ \forall j \in [i{:}i+b-1]$ and no healing period contains any frame from x_i to x_{i+b-1}.

Moreover, if $m_j = 0$ for all $j \in [k{:}n]$, all consecutive windows of size b starting within x_k to $x_{n-r-a-1}$ which do not intersect any healing period are labelled as normal. The labelling described above ensures that after the last misbehavior in a sequence, *h healing images* are ignored (i.e., not labeled) before another anomaly or normal window is defined. The value of h must be fine-tuned appropriately so that the car is ensured back safely on the road

Table 1: Evaluation results for all variants of SelfOracle across all SDCs.

			$\epsilon = 0.05, 1 - \epsilon = 0.95$									$\epsilon = 0.01, 1 - \epsilon = 0.99$								
			Unexpected								Nominal	*Unexpected*								Nominal
	AUC-PRC↑	AUC-ROC↑	TP	FP	TN	FN	TPR↑	FPR↓	F1	Prec.	FPR↓	TP	FP	TN	FN	TPR↑	FPR↓	F1	Prec.	FPR↓
DAVE-2																				
VAE	0.354	0.902	149	304	1,970	47	0.760	0.134	0.459	0.329	0.042	107	183	2,959	89	0.546	0.058	0.440	0.369	0.003
DAE	0.330	0.891	104	210	2,679	92	0.531	0.073	0.408	0.331	0.041	21	55	4,063	175	0.107	0.013	0.154	0.276	0.010
SAE	0.336	0.891	138	260	1,877	58	0.704	0.122	0.465	0.347	0.050	108	183	2,665	88	0.551	0.064	0.444	0.371	0.002
CAE	0.290	0.821	6	22	4,208	190	0.031	0.005	0.054	0.214	0.024	0	0	4,282	196	0	0	n.a.	n.a.	0
LSTM	0.357	0.903	16	34	3,990	177	0.083	0.008	0.132	0.320	0	7	12	4,119	186	0.036	0.003	0.066	0.368	0
DeepRoad	0.198	0.780	65	344	3,170	131	0.332	0.098	0.215	0.159	0.054	44	250	3,651	152	0.225	0.064	0.180	0.150	0.037
Epoch																				
VAE	0.391	0.904	158	331	1,952	51	0.756	0.145	0.453	0.323	0.049	106	169	2,858	103	0.507	0.056	0.438	0.386	0.001
DAE	0.399	0.895	112	188	2,720	97	0.536	0.065	0.440	0.373	0.042	24	34	3,653	185	0.115	0.009	0.180	0.414	0.010
SAE	0.386	0.883	147	284	2,026	62	0.703	0.123	0.459	0.341	0.050	120	175	2,838	89	0.574	0.058	0.476	0.407	0.002
CAE	0.310	0.822	6	23	3,661	203	0.029	0.006	0.050	0.207	0.020	0	0	3,731	209	0	0	n.a.	n.a.	0
LSTM	0.385	0.879	23	34	3,503	175	0.116	0.010	0.180	0.404	0.001	7	13	3,592	191	0.035	0.004	0.064	0.350	0
DeepRoad	0.213	0.807	70	308	2,917	139	0.335	0.096	0.239	0.185	0.053	43	201	3,240	166	0.206	0.058	0.190	0.176	0.040
Chauffeur																				
VAE	0.242	0.951	98	392	3,700	23	0.810	0.096	0.321	0.200	0.049	81	267	5,391	40	0.669	0.047	0.345	0.233	0.002
DAE	0.203	0.944	78	281	5,045	43	0.645	0.053	0.325	0.217	0.051	13	95	7,730	108	0.107	0.012	0.114	0.120	0.009
SAE	0.241	0.931	96	354	3,650	25	0.793	0.088	0.336	0.213	0.056	86	240	5,177	35	0.711	0.044	0.385	0.264	0.003
CAE	0.172	0.909	7	34	8,127	114	0.058	0.004	0.086	0.171	0.023	0	0	8,229	121	0	0	n.a.	n.a.	0
LSTM	0.240	0.945	11	41	7,879	111	0.090	0.005	0.126	0.212	0	4	12	8,035	118	0.033	0.002	0.058	0.250	0
DeepRoad	0.098	0.797	37	594	5,564	84	0.306	0.097	0.098	0.059	0.055	23	458	6,551	98	0.190	0.065	0.076	0.048	0.042
Totals																				
VAE	0.320	0.924	405	1027	7,622	121	0.770	0.119	0.414	0.283	0.046	294	619	11,208	232	0.559	0.052	0.409	0.322	0.002
DAE	0.301	0.911	294	679	10,444	232	0.559	0.061	0.392	0.302	0.045	58	184	15,446	468	0.110	0.012	0.151	0.240	0.009
SAE	0.312	0.907	381	898	7,553	145	0.724	0.106	0.422	0.298	0.052	314	598	10,680	212	0.597	0.053	0.437	0.344	0.002
CAE	0.255	0.864	19	79	15,996	507	0.036	0.005	0.061	0.194	0.022	0	0	16,242	526	0	0	n.a.	n.a.	0
LSTM	0.329	0.915	50	109	15,372	463	0.098	0.007	0.149	0.315	0	18	37	15,746	495	0.035	0.002	0.063	0.327	0
DeepRoad	0.159	0.799	172	1246	11,651	354	0.327	0.097	0.177	0.121	0.054	110	909	13,442	416	0.209	0.063	0.142	0.108	0.040

when the next windows are labelled. Furthermore, r images occur in between an anomaly window in which the system is supposed to predict the upcoming misbehavior, and the actual misbehavior. This period would, in practice, be used by the self-healing system to execute countermeasures against the predicted future misbehavior. Intuitively, misbehavior prediction is expected to be much harder as the value of r increases. In our experiments, we set the value of $n = b = 30$ frames (i.e., normal/anomalous windows), which is ≈3s.[3] The size of the healing window was set to $h = 60$ (> 5s) frames, and the size of the reaction window to $r = 50$ (> 4s) frames.

5.5.2 Metrics used for Analysis. If the loss score for an image is higher than the automatically estimated threshold θ (Section 4.2), SelfOracle triggers an alarm. Consequently, a true positive is defined when SelfOracle triggers an alarm during an anomalous window, early enough to predict a misbehavior. Conversely, a false negative occurs when SelfOracle does not trigger an alarm during an anomalous window, thus failing at predicting a misbehaviour in time for triggering self-healing. A false positive represents a false alarm by SelfOracle, whereas true negative cases occur when SelfOracle detects correct detection of normality.

We assume that a single alarm immediately starts the self healing system, such that multiple consecutive alarms within the healing time are ignored. Correspondingly, once a FP occurs and the self healing system is running, additional consecutive FP windows have no effect in practice, and are thus excluded from our analysis.

[3]In our setting, Udacity frame rate was approximately 10/13 fps.

Our goal is to achieve high recall, or true positive rate (TPR, defined as TP/TP+FN), i.e., true alarms, while minimizing the complement of specificity, or false positive rate (FPR, defined as FP/TN+FP), i.e., labelling safe situations as unsafe. We are also interested in the F1-score ($F_1 = 2 \cdot \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$) because, in practice, it is informative to have a high F1-score at a given threshold.

We also consider two threshold-independent metrics for evaluating classifiers at various thresholds such as AUC-ROC (area under the curve of the Receiver Operating Characteristics), and AUC-PRC (area under the Precision-Recall curve). We included AUC-PRC because AUC-ROC may be not indicative when data are heavily unbalanced, which is our case, since anomalies are rare [16, 35].

5.6 Results

Effectiveness (RQ$_1$). Table 1 presents the effectiveness results on a per-SDC model basis. Columns 2 and 3 show threshold-independent measures of AUC-PRC and AUC-ROC. The remaining of the table shows the effectiveness metrics across two confidence thresholds that we found interesting for analysis and correspond to $\epsilon = 0.05$ and $\epsilon = 0.01$ (at lower values of ϵ, both FPR and TPR get close to zero, making the misbehaviour predictor useless).

Overall, LSTM and VAE are the best performing reconstructors on the AUC-PRC and AUC-ROC metrics. Columns 12 and 21 (Nominal/FPR) show FPR under conditions similar to those of the training set. Values are almost always near to zero and occasionally even equal to zero (this is indicated by omitting the decimals) across the variants of SelfOracle. This is an empirical validation of the accuracy of the Gamma distribution as a statistical model for the reconstruction errors. In fact, at $\epsilon = 0.05$, most FPR reported in column 12 are very close to the theoretical value (see, e.g., rows under *Totals*). At $\epsilon = 0.01$, some values drop to zero. This means that in those configurations SelfOracle will raise no false alarm when the SDC drives in nominal conditions. For instance, with both thresholds, LSTM never raised false alarms within the 23,728 normal windows.

In terms of TPR (to be maximized) and FPR (to be minimized), the best reconstructors are VAE and SAE, with comparable overall performance: 77/11% (TPR/FPR of VAE) and 72/10% (SAE) at $\epsilon = 0.05$; 55%/5% (VAE) and 59/5% (SAE) at $\epsilon = 0.01$. It can be noticed that FPR is higher than ϵ (10% vs 5% and 5% vs 1%) with both reconstructors. This was expected, since we are measuring FPR in tracks with injected anomalies. These tracks may differ substantially from the nominal tracks when anomalies are injected with low intensities. These conditions are often not so extreme to cause a misbehaviour, which explains why the FPR under anomalous conditions is over-approximated. However, it is important to notice that even in such non nominal conditions, in most cases the FPR remains low and not too far from the theoretical prediction ϵ.

To answer RQ$_1$, reconstructors VAE and SAE achieve a FPR in nominal conditions close to the theoretical expectations (respectively 5% and 1%, in the two considered configurations), that in anomalous conditions FPR increases by a moderate amount (respectively, +5% and +4%). The achieved TPR is quite high (with VAE, SAE, respectively 77/72% and 55/59%). The relation between area difference d and precision/recall delta is quadratic, thus precision/recall improvements are expected to be the order of $\sqrt(d)$. For instance, in Table 1 (SAE vs DAE, DAVE-2), we have $d = 0.006$; for the specific threshold that ensures $\epsilon = 0.05$, the corresponding improvements for TPR and FPR are 0.17 and 0.13, respectively.

Prediction (RQ$_2$). Figure 8 shows the AUC-PRC of the various configurations of SelfOracle over different reaction periods. The general trend is that predictions get harder when the SDC is far from a critical scenario, having a longer reaction period to prevent the misbehavior, but quite surprisingly there is no drop in performance as we move away from the misbehaviour. Our explanation of this unexpected finding is that the tracks used for the evaluation of the approach contain always a relatively high degree of anomalous features, which might trigger a self-healing reaction. Occasionally, the level of detected anomalies surpasses the threshold and the misbehaviour predictor raises an alarm. Correspondingly, although slightly reduced, the signals of an upcoming misbehaviour exist in images quite far (even 60 frames, around 6 s) from the misbehaviour.

To answer RQ$_2$, the performance of SelfOracle degrades smoothly as we anticipate the prediction (AUC-PRC remains quite high up to 6 s before the misbehaviour). In our experiments, SelfOracle is not sensitive to the choice of k if chosen in the range [4..W/2] (W=normal window size). We remark that this result must be taken with care and may be partially due to the characteristics of the considered tracks, which contain a continuously and smoothly increasing degree of injected anomalies by design.

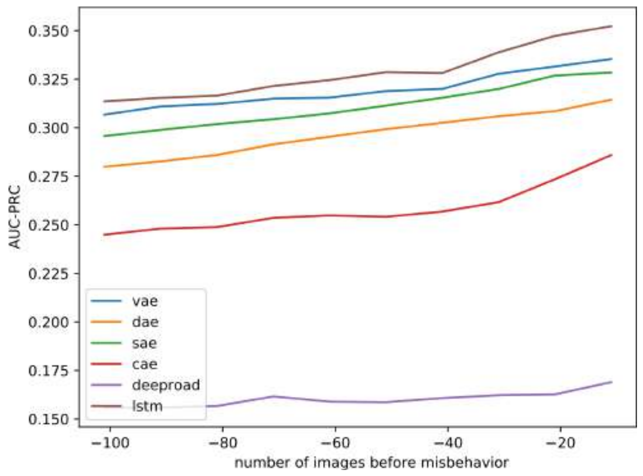

Figure 8: Misbehavior prediction capability over time.

Comparison (RQ$_3$). In our experiments, SelfOracle is constantly superior to DeepRoad at predicting misbehaviours. Results of AUC-PRC and AUC-ROC show significant improvements across all thresholds, regardless of the technique being used and the reaction period considered (in Figure 8, DeepRoad is the lowest curve). With $\epsilon = 0.05$ (resp. $\epsilon = 0.01$), VAE and SAE (see Table 1) predict correctly more than twice the misbehaviours exposed by DeepRoad, with a TPR = 77/72% vs 32% (respectively, 55/59% vs 20%), at comparable false positive rate FPR = 11/10% vs 9% (respectively, 5% vs 6%).

Concerning the runtime, in our experiments, the autoencoders took ≈3 ms per prediction whereas DeepRoad took ≈45 ms per prediction (+1400% increment). While such runtime measures seem acceptable in practical scenarios, it is worth remembering that DeepRoad required us to dramatically sub-sample the training set available for our experiments. SDC manufacturers use much larger training datasets than the one used for our empirical study and DeepRoad's runtime is quite sensitive to the size of the training set, which makes it quite unsuitable for online misbehaviour prediction because both computationally very expensive. On the contrary, autoencoders are a promising option, given their relatively simple architecture. In particular, SAE is very efficient, yet it has comparable performance as the more sophisticated VAE.

To answer RQ$_3$, SelfOracle outperforms DeepRoad in all respects: computational cost, accuracy of misbehaviour prediction (see TPR), and minimization of false alarms (see FPR and AUC-PRC).

5.7 Threats to Validity

Internal validity. We compared all variants of SelfOracle and DeepRoad under identical parameter settings, and on the same evaluation set. The main threat to internal validity concerns our custom implementation of unexpected conditions within the simulator. However, this was a mandatory choice, since we are not aware of open-source driving simulators that can inject unexpected execution contexts in a controllable way. Another possible threat may be the choice and the training of our own SDCs, which may exhibit a large number of misbehaviours if trained inadequately. We mitigated this threat by training and fine-tuning the best publicly available driving models. Our own implementation of DeepRoad may be another threat to internal validity, that we mitigated by developing an implementation which improves the original one by processing 5× more information.

External validity. We used a limited number of self-driving systems in our evaluation, as well as tracks, which pose a threat in terms of generalizability of our results. We tried to mitigate this threat by choosing popular real-world SDC models which achieved competitive scores in the Udacity challenge.

Reproducibility. All our results, the source code of SelfOracle, the simulator, and all subjects are available [45].

6 RELATED WORK

Autoencoder-based Anomaly Detection. A major testing issue with machine-learning based systems is finding a confidence/uncertainty metric of the machine-learning component, and being able to distinguish the nominal vs abnormal behaviours of the system in operation. To this extent, the machine learning literature has seen a proliferation of autoencoder-based anomaly detection techniques applied to different domains, among which time series [27], surveillance videos [15], robot-assisted feeding [31], or web applications [54]. All such papers propose and evaluate their techniques mostly for classification problems, on which the concept of misbehaviour is more easily tractable. Differently, we target online misbehaviour prediction for autonomous driving systems, i.e., complex DNNs that predict the car's actuators values. Our work aims to improve the dependability of a whole autonomous driving system, and prevent the occurrence of future failures, rather than focusing on individual DNN's model-level inaccuracies.

Concerning the very definition of unseen situation, Bolte et al. [10] provide a definition of misbehaviour based on non-predictable relevant objects in a relevant location around the car. Such a definition can be used in our framework by replacing our reconstruction error component with an object recognition component. Several works leverage anomaly detection techniques to identify unexpected execution contexts during the system's operation [4, 10, 21, 33, 53, 55], whereas a few papers are related to online risk assessment and failure probability estimation for MLS [39, 50]. In the context of autonomous driving systems, Henriksson et al. [21] use the negative of the log likelihood of the images generated by a VAE as an anomaly score for driving images. However, their evaluation is performed on unrealistic scenarios because the data distribution of the images in the test set (urban scenes) is by far quite different from that of the training set (highway scenes). In our experiment, we create the anomalous set by gradually injecting anomalous conditions starting from the training set tracks, which allows a more realistic transition from nominal to unexpected scenarios.

Adversarial Input Generation. Adversarial input generation approaches aim at generating inputs that trigger inconsistencies between multiple autonomous driving systems [34], or between the original and transformed driving scenarios [30, 44, 55]. These works exploit the well-known fragility of DNNs to adversarial examples. Therefore, their main use case concerns the identification of underrepresented scenarios in the training data (e.g., snowy weather condition) to support re-training and better generalization after re-training. The only comparable technique is the online input validation of DeepRoad [55], for which we carried out an explicit comparison in our empirical study, finding poor performance when used for online misbehaviour prediction.

Despite the different goal (test generation vs misbehaviour prediction), we share with these works the problem of how to empirically validate the proposed technique in the absence of a precise oracle that defines the expected behaviour of a self-driving car. The prevalent choice in test generators [30, 34, 44, 55] is to address the oracle problem by *differential testing*, i.e., by comparing the behaviours of multiple DNNs, or by *metamorphic testing*, i.e., by comparing the behaviour before and after applying a metamorphic transformation to the input. Approaches based on verification are also being under investigation [19]. In this paper, we adopt a precise definition of *DNN misbehaviour*, which gives us a very accurate *functional oracle*, with no need for differential testing, metamorphic testing, or verification.

Search-based Test Generation. Abdessalem et al. [1, 6, 7] combine genetic algorithms and machine learning to test a pedestrian detection system. Mullins et al. [29] use Gaussian processes to drive the search towards yet unexplored regions of the input space, whereas Gambi et al. [18] propose AsFault, a search-based test generator for autonomous vehicles based on procedural content generation. AsFault uses search operators which mutate and recombine road segments to construct road networks for testing the lane keeping functionality of self-driving cars. Their goal is to generate extreme and challenging roads, maximizing the number of observed OBEs, while our goal is to predict OBEs.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we studied the problem of estimating the confidence of the DNN-based autonomous in response to unexpected execution contexts. Our tool SelfOracle was able to anticipate by several seconds many potentially safety-critical misbehaviours, such as out-of-bound episodes or collisions, with a low false alarm rate, outperforming the input validator of DeepRoad.

Future work concerns experimenting with other white-box DNN confidence metrics, as well as other types of reconstructors, among which denoising autoencoders [36], and adversarial autoencoders [5]. On the same line, more elaborate LSTM-based solutions other than the simple architecture proposed in this paper will be investigated. It would be also interesting to characterize and predict other kinds of misbehaviours (e.g., derivative of steering angle) in order to allow confidence-guided self-healing within the simulator, and leverage more sensor information sources (e.g., LIDAR) to improve SelfOracle's prediction accuracy.

We believe that our promising results in online misbehaviour detection, united with the availability of a labeled dataset of crashes and a simulation environment, can foster novel approaches for online prediction and self-healing of autonomous driving systems.

ACKNOWLEDGMENTS

This work was partially supported by the H2020 project PRECRIME, funded under the ERC Advanced Grant 2017 Program (ERC Grant Agreement n. 787703).

REFERENCES

[1] Raja Ben Abdessalem, Annibale Panichella, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. 2018. Testing Autonomous Cars for Feature Interaction Failures Using Many-objective Search. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE 2018)*. ACM, New York, NY, USA, 143–154. https://doi.org/10.1145/3238147.3238192

[2] David Alvarez-Melis and Tommi S. Jaakkola. 2018. Towards Robust Interpretability with Self-Explaining Neural Networks. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*. 7786–7795.

[3] Jinwon An and Sungzoon Cho. 2015. Variational Autoencoder based Anomaly Detection using Reconstruction Probability.

[4] Adina Aniculaesei, Jörg Grieser, Andreas Rausch, Karina Rehfeldt, and Tim Warnecke. 2018. Towards a holistic software systems engineering approach for dependable autonomous systems. In *Proceedings of the 1st International Workshop on Software Engineering for AI in Autonomous Systems - SEFAIS*. ACM Press. https://doi.org/10.1145/3194085.3194091

[5] Laura Beggel, Michael Pfeiffer, and Bernd Bischl. 2019. Robust Anomaly Detection in Images using Adversarial Autoencoders. *CoRR* abs/1901.06355 (2019). arXiv:1901.06355 http://arxiv.org/abs/1901.06355

[6] R. Ben Abdessalem, S. Nejati, L. C. Briand, and T. Stifter. 2016. Testing advanced driver assistance systems using multi-objective search and neural networks. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 63–74.

[7] R. Ben Abdessalem, S. Nejati, L. C. Briand, and T. Stifter. 2018. Testing Vision-Based Control Systems Using Learnable Evolutionary Algorithms. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. 1016–1026. https://doi.org/10.1145/3180155.3180160

[8] BGR Media, LLC. 2018. Waymo's self-driving cars hit 10 million miles. https://techcrunch.com/2018/10/10/waymos-self-driving-cars-hit-10-million-miles. Online; accessed 18 August 2019.

[9] 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). http://arxiv.org/abs/1604.07316

[10] J. Bolte, A. Bar, D. Lipinski, and T. Fingscheidt. 2019. Towards Corner Case Detection for Autonomous Driving. In *2019 IEEE Intelligent Vehicles Symposium (IV)*. 438–445. https://doi.org/10.1109/IVS.2019.8813817

[11] Georg Burkhard, S. Vos, N. Munzinger, E. Enders, and D. Schramm. 2018. Requirements on driving dynamics in autonomous driving with regard to motion and comfort. In *18. Internationales Stuttgarter Symposium*, Michael Bargende, Hans-Christian Reuss, and Jochen Wiedemann (Eds.). Springer Fachmedien Wiesbaden, Wiesbaden, 683–697.

[12] Guilherme O. Campos, Arthur Zimek, Jörg Sander, Ricardo J. Campello, Barbora Micenková, Erich Schubert, Ira Assent, and Michael E. Houle. 2016. On the Evaluation of Unsupervised Outlier Detection: Measures, Datasets, and an Empirical Study. *Data Min. Knowl. Discov.* 30, 4 (July 2016), 891–927. https://doi.org/10.1007/s10618-015-0444-8

[13] Vinton G. Cerf. 2018. A Comprehensive Self-driving Car Test. *Commun. ACM* 61, 2 (Jan. 2018), 7–7. https://doi.org/10.1145/3177753

[14] S. C. Choi and R. Wette. 1969. Maximum Likelihood Estimation of the Parameters of the Gamma Distribution and Their Bias. *Technometrics* 11, 4 (1969), 683–690. https://doi.org/10.1080/00401706.1969.10490731

[15] Yong Shean Chong and Yong Haur Tay. 2017. Abnormal Event Detection in Videos Using Spatiotemporal Autoencoder. In *Advances in Neural Networks - ISNN 2017*, Fengyu Cong, Andrew Leung, and Qinglai Wei (Eds.). Springer International Publishing, Cham, 189–196.

[16] Jesse Davis and Mark Goadrich. 2006. The relationship between Precision-Recall and ROC curves. In *Proceedings of the 23rd international conference on Machine learning - ICML06*. ACM Press. https://doi.org/10.1145/1143844.1143874

[17] Electrek. 2016. Tesla Model S driver crashes into a van while on Autopilot. https://electrek.co/2016/05/26/tesla-model-s-crash-autopilot-video/. Online; accessed 18 August 2019.

[18] Alessio Gambi, Marc Mueller, and Gordon Fraser. 2019. Automatically Testing Self-driving Cars with Search-based Procedural Content Generation. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2019)*. ACM, New York, NY, USA, 318–328. https://doi.org/10.1145/3293882.3330566

[19] Divya Gopinath, Guy Katz, Corina S. Păsăreanu, and Clark Barrett. 2018. DeepSafe: A Data-Driven Approach for Assessing Robustness of Neural Networks. In *Automated Technology for Verification and Analysis*, Shuvendu K. Lahiri and Chao Wang (Eds.). Springer International Publishing, Cham, 3–19.

[20] Fitash Ul Haq, Donghwan Shin, Shiva Nejati, and Lionel Briand. 2020. Comparing Offline and Online Testing of Deep Neural Networks: An Autonomous Car Case Study. In *Proceedings of 13th IEEE International Conference on Software Testing, Verification and Validation (ICST '20)*. IEEE.

[21] Jens Henriksson, Christian Berger, Markus Borg, Lars Tornberg, Cristofer Englund, Sankar Raman Sathyamoorthy, and Stig Ursing. 2019. Towards Structured Evaluation of Deep Neural Network Supervisors. In *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*. IEEE. https://doi.org/10.1109/aitest.2019.00-12

[22] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (Nov. 1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[23] Paul Hoel. 1984. *Introduction to Mathematical Statistics*. John Wiley.

[24] keras. [n. d.]. Building Autoencoders in Keras. https://blog.keras.io/building-autoencoders-in-keras.html. Online; accessed 21 August 2019.

[25] Keras. [n. d.]. VGG19. https://keras.io/applications/#vgg19/. Online; accessed 21 August 2019.

[26] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding Deep Learning System Testing Using Surprise Adequacy. In *Proceedings of the 41st International Conference on Software Engineering (ICSE '19)*. IEEE Press, Piscataway, NJ, USA, 1039–1049. https://doi.org/10.1109/ICSE.2019.00108

[27] Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. 2016. LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection. *CoRR* abs/1607.00148 (2016). arXiv:1607.00148 http://arxiv.org/abs/1607.00148

[28] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. 2011. Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction. In *Artificial Neural Networks and Machine Learning – ICANN 2011*, Timo Honkela, Włodzisław Duch, Mark Girolami, and Samuel Kaski (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 52–59.

[29] Galen E. Mullins, Paul G. Stankiewicz, R. Chad Hawthorne, and Satyandra K. Gupta. 2018. Adaptive generation of challenging scenarios for testing and evaluation of autonomous vehicles. *Journal of Systems and Software* 137 (2018), 197–215. https://doi.org/10.1016/j.jss.2017.10.031

[30] S. Müller, D. Hospach, O. Bringmann, J. Gerlach, and W. Rosenstiel. 2015. Robustness Evaluation and Improvement for Vision-Based Advanced Driver Assistance Systems. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*. 2659–2664. https://doi.org/10.1109/ITSC.2015.427

[31] D. Park, Y. Hoshi, and C. C. Kemp. 2018. A Multimodal Anomaly Detector for Robot-Assisted Feeding Using an LSTM-Based Variational Autoencoder. *IEEE Robotics and Automation Letters* 3, 3 (July 2018), 1544–1551.

[32] particle-system 2019. Unity3d Particle System. https://docs.unity3d.com/ScriptReference/ParticleSystem.html.

[33] Naman Patel, Apoorva Nandini Saridena, Anna Choromanska, Prashanth Krishnamurthy, and Farshad Khorrami. 2018. Adversarial Learning-Based On-Line Anomaly Monitoring for Assured Autonomy. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain, October 1-5, 2018*. 6149–6154. https://doi.org/10.1109/IROS.2018.8593375

[34] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*. ACM, New York, NY, USA, 1–18. https://doi.org/10.1145/3132747.3132785

[35] Takaya Saito and Marc Rehmsmeier. 2015. The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets. *PLOS ONE* 10, 3 (March 2015), e0118432. https://doi.org/10.1371/journal.pone.0118432

[36] Mayu Sakurada and Takehisa Yairi. 2014. Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction. In *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis (MLSDA'14)*. Association for Computing Machinery, New York, NY, USA, 4–11. https://doi.org/10.1145/2689746.2689747

[37] Claude Elwood Shannon. 1948. A Mathematical Theory of Communication. *The Bell System Technical Journal* 27, 3 (7 1948), 379–423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x

[38] Karen Simonyan and Andrew Zisserman. [n. d.]. Very Deep Convolutional Networks for Large-Scale Image Recognition. ([n. d.]). arXiv:cs.CV/1409.1556v6 VGGNet, https://gist.github.com/baraldilorenzo/07d7802847aaad0a35d3.

[39] Mark Strickland, Georgios Fainekos, and Hani Ben Amor. 2018. Deep predictive models for collision risk assessment in autonomous driving. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018 (Proceedings - IEEE International Conference on Robotics and Automation)*. Institute of Electrical and Electronics Engineers Inc., 4685–4692. https://doi.org/10.1109/ICRA.2018.8461160

[40] Team Chauffeur. 2016. Steering angle model: Chauffeur. https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models/chauffeur. Online; accessed 18 August 2019.

[41] Team Epoch. 2016. Steering angle model: Epoch. https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models/cg23. Online; accessed 18 August 2019.

[42] The Verge. 2016. A Google self-driving car caused a crash for the first time. https://www.theverge.com/2016/2/29/11134344/google-self-driving-car-crash-report. Online; accessed 18 August 2019.

[43] The Verge. 2019. Tesla hit with another lawsuit over a fatal Autopilot crash. https://www.theverge.com/2019/8/1/20750715/tesla-autopilot-crash-lawsuit-wrongful-death. Online; accessed 18 August 2019.

[44] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. DeepTest: Automated Testing of Deep-neural-network-driven Autonomous Cars. In *Proceedings of the 40th International Conference on Software Engineering (ICSE '18)*. ACM, New York, NY, USA, 303–314. https://doi.org/10.1145/3180155.3180220

[45] SelfOracle 2020. Misbehaviour Prediction for Autonomous Driving Systems. https://github.com/testingautomated-usi/selforacle/.

[46] track3 2019. Unity3D Snow Mountain Track. https://assetstore.unity.com/packages/3d/environments/roadways/mountain-race-track-53775.

[47] Udacity. 2017. A self-driving car simulator built with Unity. https://github.com/udacity/self-driving-car-sim. Online; accessed 18 August 2019.

[48] Udacity. 2017. Udacity self-driving car's challenge. https://github.com/udacity/self-driving-car/. Online; accessed 18 August 2019.

[49] Udacity. 2017. Udacity self-driving car's datasets. https://github.com/udacity/self-driving-car/tree/master/datasets. Online; accessed 18 August 2019.

[50] Jonathan Uesato, Ananya Kumar, Csaba Szepesvári, Tom Erez, Avraham Ruderman, Keith Anderson, Krishnamurthy (Dj) Dvijotham, Nicolas Heess, and Pushmeet Kohli. 2019. Rigorous Agent Evaluation: An Adversarial Approach to Uncover Catastrophic Failures. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

[51] unity 2019. Unity3D. https://unity.com.

[52] V. T. Vasudevan, A. Sethy, and A. R. Ghias. 2019. Towards Better Confidence Estimation for Neural Models. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 7335–7339. https://doi.org/10.1109/ICASSP.2019.8683359

[53] 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 '19)*. IEEE Press, Piscataway, NJ, USA, 1245–1256. https://doi.org/10.1109/ICSE.2019.00126

[54] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, et al. 2018. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 2018 World Wide Web Conference*. 187–196.

[55] 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 *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE 2018)*. ACM, New York, NY, USA, 132–142. https://doi.org/10.1145/3238147.3238187