# TigAug: Data Augmentation for Testing Traffic Light Detection in Autonomous Driving Systems

Anonymous Author(s)

## ABSTRACT

Autonomous vehicle technology has been vigorously developed in the last decades with recent advances in sensing and computing technology. There is an urgent need to ensure the reliability and robustness of autonomous driving systems (ADSs). Despite the recent achievements in automatically testing various modules of ADSs, little attention has been paid on the automated testing of traffic light detection models in ADSs. A common practice is to manually collect and label traffic light data. However, it is labor-intensive, and even impossible to collect diverse data under different driving environments.

To address these problems, we propose and implement TigAug to automatically augment labeled traffic light images for testing traffic light detection models in ADSs. We construct two families of metamorphic relations and three families of transformations based on a systematic understanding of weather environments, camera properties, and traffic light properties. We use augmented images to detect erroneous behaviors of traffic light detection models by transformation-specific metamorphic relations, and to improve the performance of traffic light detection models by retraining. Large-scale experiments with four state-of-the-art traffic light detection models and two traffic light datasets have demonstrated that i) TigAug is effective in testing traffic light detection models, ii) TigAug is efficient in synthesizing traffic light images and retraining models, and iii) TigAug generates traffic light images with acceptable naturalness.

## 1 INTRODUCTION

Academic and industrial efforts have been increasingly devoted to vigorously developing autonomous vehicle technology in the last decades. These developments have been fueled by recent advances in sensing and computing technology together with the impact on automotive transportation and the benefit to society (e.g., reducing vehicle collisions, providing personal mobility to disabled people, and reducing ill effects of driving stress) [41]. The automation system of autonomous vehicles, also known as autonomous driving system (ADS), is typically organized into two main parts, i.e., the perception system and the decision making system [6]. The perception system estimates the vehicle and environment state using the data captured by on-board sensors such as camera, LIDAR, RADAR and GPS, while the decision making system navigates the vehicle from its initial position to the final destination specified by the user.

As a safety-critical system, it is important to ensure the reliability and robustness of an ADS. Unfortunately, the state-of-the-practice ADSs from leading companies such as Tesla, Waymo and Uber are still vulnerable to corner cases and exhibit incorrect behaviors, due to the extremely complicated and diverse real-world driving environments. These incorrect behaviors might lead to catastrophic consequences and unsustainable losses, as evidenced by many reported traffic accidents [5, 27, 39]. Therefore, on-road testing is adopted by these leading companies to achieve quality assurance for ADSs. To further test extreme conditions or corner cases that are difficult or expensive to produce in real-world environments, simulation testing is also widely adopted by these leading companies [24, 26].

In recent years, many testing approaches have been developed by the software engineering community to ensure the reliability and robustness of ADSs. Specifically, one main line of work attempts to apply search-based testing to detect safety violations for ADSs [1, 2, 9, 17, 18, 21, 32, 50, 54, 60]. They formulate a test scenario as a vector of variables (e.g., vehicle speed and fog degree), and apply generic algorithms to search for test scenarios that violate safety requirements. Another main thread of work focuses on testing DNN (deep neural network)-based modules in ADSs. They use metamorphic testing to generate images of driving scenes [43, 49, 56, 57] and point clouds of driving scenes [20]. For example, DeepTest [56] uses weather transformations (e.g., adding rain effect) and affine transformations (e.g., translation) to synthesize images for testing steering angle decision models in ADSs. Similarly, LiRTest [20] applies weather transformations and affine transformations to synthesize point clouds for testing 3D object detection models in ADSs.

Despite these advances in finding erroneous behaviors in various modules of ADSs, little attention has been paid on the testing of traffic light detection models in ADSs. Traffic lights are used to control the movement of traffic, and thus play an important role in ensuring traffic safety. Therefore, ADSs employ traffic light detection models (e.g., *YOLO* [45], *Faster R-CNN* [46], and *SSD* [35]) to detect the position of one or more traffic lights in the driving scene (e.g., represented in an image) and recognize their states (e.g., red, green, and yellow) [6]. When an ADS fails to correctly recognize traffic lights, it may cause serious traffic accidents. For example, Tesla's Autopilot misidentified the moon as a yellow light, causing the vehicle to unexpectedly slow down at highway speeds [31]. Uber's autonomous vehicle passed through a red light three seconds after the light had turned red and while a pedestrian was in the crosswalk [4]. Therefore, it is crucial to specifically test traffic light detection in ADSs.

The testing of traffic light detection heavily relies on the labeled traffic light data (i.e., images of traffic lights), which is usually manually collected. Specifically, on-road testing is employed to capture images of traffic lights via cameras. However, it is resource-intensive or even impossible to collect diverse data under different driving environments. Then, the captured traffic light images are manually labeled to mark the position and state of traffic lights. However, it is a labor-intensive and time-consuming task, especially when the number of traffic light images increases. To the best of our knowledge, Bai et al.'s work [7] is the first and only work to automatically generate synthetic images of traffic lights. However, they only consider one transformation scenario, i.e., changing the color of traffic lights, hindering the diversity of generated traffic light images and hurting the effectiveness of traffic light detection testing.

To address these problems, we propose and implement a systematic data augmentation approach, named TigAug, to automatically augment labeled traffic light images for testing traffic light detection

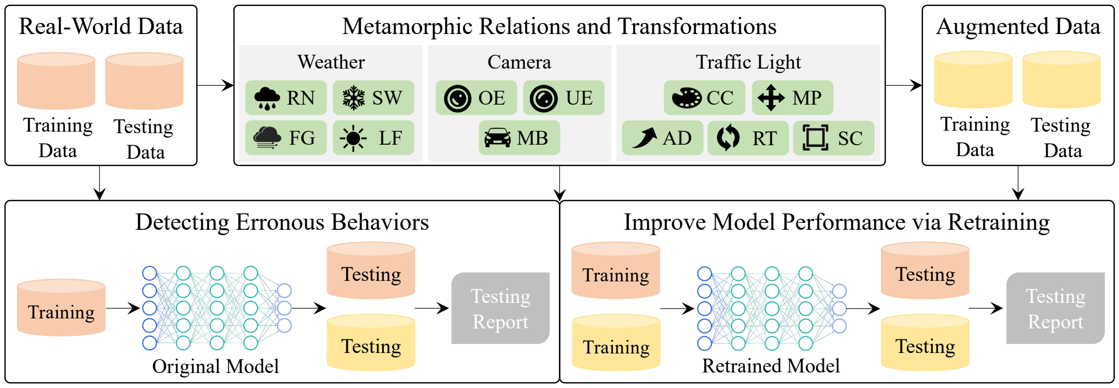


**Figure 1: Approach Overview of TIGAUG**

models in ADSs. Specifically, we systematically construct two families of metamorphic relations and three families of transformations based on a systematic understanding of weather environments, camera properties, and traffic light properties. Given a labeled traffic light image from real world, TIGAUG applies transformations to synthesize augmented traffic light images. Then, it uses transformation-specific metamorphic relations between the real-world image and those augmented images to identify erroneous behaviors of traffic light detection models in ADSs. Moreover, the augmented images can be used to retrain and improve traffic light detection models.

We conduct large-scale experiments to evaluate TIGAUG, using four state-of-the-art traffic light detection models and two traffic light datasets. First, we apply transformations on the testing data, and the mean average precision of the original models suffers a decrease of 40.5% on the augmented testing data. Second, we obtain retrained models by adding augmented training data, and the mean average precision of the retrained models achieves an increase of 50.7% on the augmented testing data. These results demonstrate that TIGAUG is effective in detecting erroneous behaviors and improving the performance of traffic light detection models in ADSs. Third, we measure the time overhead of data augmentation and retraining to evaluate the efficiency of TIGAUG, and it takes on average 0.88 seconds to synthesize an image and 14 hours to retrain a model, which is acceptable given the improved mean average precision. Finally, we manually identify unnatural images from the synthesized images, and 27.9% of the synthesized images from each transformation are considered as unnatural, but they have little impact on model performance. This result indicates that our synthesized traffic light images can be directly used without manual cleaning.

In summary, this work makes the following contributions.

- We constructed two families of metamorphic relations and three families of transformations to simulate the impact of weather environments, camera properties and traffic light properties on the captured traffic light images in real-world driving environments.
- We implemented the proposed approach as a prototype tool, named TIGAUG, to automatically augment traffic light images for detecting erroneous behaviors and improving the performance of traffic light detection models in ADSs.
- We conducted large-scale experiments, using four state-of-the-art traffic light detection models and two traffic light datasets, to demonstrate the effectiveness and efficiency of TIGAUG.

## 2 METHODOLOGY

We design and implement TIGAUG as a systematic data augmentation approach to automatically augment labeled traffic light images for testing traffic light detection models in ADSs. Fig. 1 presents an approach overview of TIGAUG. It is built upon our domain understanding of how weather environments, camera properties, and traffic light properties affect the traffic light images captured by cameras in real-world environments. In other words, we construct two families of metamorphic relations (see Sec. 2.1) and three families of transformations (see Sec. 2.2) with respect to weather environments, camera properties, and traffic light properties.

On the basis of our domain analysis, TIGAUG first applies transformations on labeled traffic light data from real world (i.e., real-world data) to generate augmented traffic light data (i.e., augmented data). Then, TIGAUG uses transformation-specific metamorphic relations between the real-world testing data and those augmented testing data to identify erroneous behaviors of traffic light detection models trained from real-world training data. Moreover, TIGAUG retrains traffic light detection models by adding augmented training data to real-world training data so as to improve model performance.

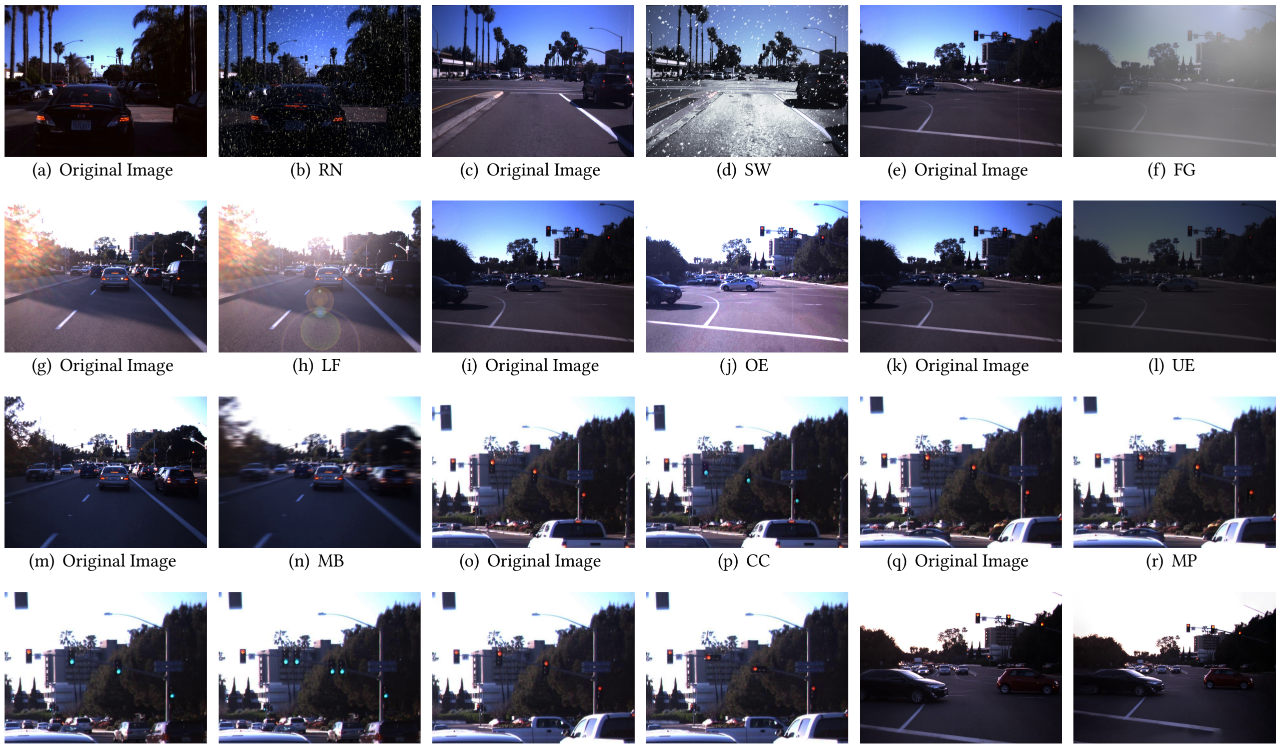
### 2.1 Metamorphic Relations

A key challenge of traffic light data augmentation is to automatically determine the expected output of traffic light detection models on the augmented data. Metamorphic relations are known to be capable of alleviating this test oracle challenge [11, 48]. In our scenario, each metamorphic relation describes a property between the outputs of traffic light detection models on the real-world traffic light data and the augmented traffic light data. Specifically, for the three families of transformations (see Sec. 2.2), we build two families of metamorphic relations. Before elaborating the metamorphic relations, we first formulate relevant notations. For a traffic light image $i$ from real-world data $\mathbb{I}$, the output of a traffic light detection model $m$ on $i$ is denoted as $m[[i]]$; and the augmented image after applying a transformation $\tau$ from weather transformations $\mathbb{W}$, camera transformations $\mathbb{C}$ or traffic light transformations $\mathbb{L}$ is denoted as $\tau(i)$.

Then, we formulate the metamorphic relation for weather and camera transformations by Eq. 1,

$$\forall\, i \in \mathbb{I} \land \forall\, \tau \in \mathbb{W} \cup \mathbb{C},\ \mathcal{E}\{m[[i]], m[[\tau(i)]]\} \tag{1}$$

where $\mathcal{E}$ is a criterion asserting the equality of detection outputs on real-world image and augmented image. This metamorphic relation



(a) Original Image (b) RN (c) Original Image (d) SW (e) Original Image (f) FG
(g) Original Image (h) LF (i) Original Image (j) OE (k) Original Image (l) UE
(m) Original Image (n) MB (o) Original Image (p) CC (q) Original Image (r) MP
(s) Original Image (t) AD (u) Original Image (v) RT (w) Original Image (x) SC

**Figure 2: Sample Traffic Light Images Synthesized by Our Transformations**

indicates that no matter how weather conditions and camera effects are synthesized into a real-world image by applying weather and camera transformations, the detection output of a traffic light detection model on the augmented image $\tau(i)$ is expected to be consistent with that on the corresponding real-world image $i$. For example, if the snow effect or the overexposure effect is synthesized into a real-world image $i$ to generate an augmented image $i'$, the position and state of traffic lights in $i$ are the same to those in $i'$, and hence the detection output on $i$ and $i'$ should be identical. Otherwise, erroneous behaviors are revealed by violating this metamorphic relation.

Similarly, we formulate the metamorphic relation for traffic light transformations by Eq. 2.

$$\forall\ i \in \mathbb{I} \land \forall\ \tau \in \mathbb{L},\ \mathcal{E}\{\tau(m[[i]]), m[[\tau(i)]]\} \tag{2}$$

This metamorphic relation indicates that as the position and state of traffic lights are changed by applying traffic light transformations, the detection output of a traffic light detection model on the augmented image $\tau(i)$ is expected to be correspondingly changed by applying the transformation $\tau$ to the detection output on the corresponding real-world image $i$. For example, if the state of a traffic light in a real-world image $i$ is changed from red to green to synthesize an augmented image $i'$, the detection output on $i'$ should be identical to that applying the same color change to the detection output on $i$.

Finally, we use mean average precision (mAP) to derive the equality criterion $\mathcal{E}$ because mAP is commonly used to evaluate object detection systems and it can compensate small drift of detected bounding boxes from the ground truth bounding box. Specifically, mAP is the mean of average precision (AP) scores of different object classes (e.g., different traffic light states in our scenario). AP is computed by the area under the precision-recall curve [15]. Therefore, AP takes into account both precision and recall. To compute precision and recall, intersection over union (IoU) is used to measure the overlap between the detected bounding box and the ground truth bounding box. If IoU is greater than a threshold $\theta$ (e.g., 0.5), the detection is classified as a true positive; otherwise, the detection is classified as a false positive. To take into account the impact of $\theta$, mAP@[.50,.95] is used to compute the average of mAP under 10 IoU thresholds from 0.50 to 0.95 with a step size of 0.05 [34]. Hereafter, we use mAP to simplify mAP@[.50,.95] for the ease of presentation.

## 2.2 Transformations

In order to obtain more traffic light data exploring the input-output spaces of traffic light detection models with less labor and time cost, we implement twelve transformations to generate synthesized data that is close to real-world data in a small amount of time. As shown in Fig. 1, these transformations are rain (RN), snow (SW), fog (FG), lens flare (LF), overexposure (OE), underexposure (UE), motion blur (MB), changing color of traffic lights (CC), moving position of traffic lights (MP), adding traffic lights (AD), rotating traffic lights (RT), and scaling traffic lights (SC). These transformation are classified into three families, i.e., weather transformations (to mimic the effects of different weather environments), camera transformations (to simulate different camera effects), and traffic light transformations (to enrich different positions and states of traffic lights).

Before we clarify the detailed definition and design of each transformation in each family, we first formulate relevant notations. For a real-world traffic light image, it contains a set of traffic lights

$T = \{t_1, t_2, ..., t_n\}$ with corresponding labels $L = \{l_1, l_2, ..., l_n\}$. The label $l$ of a traffic light $t$ is denoted as a 5-tuple $\langle x_1, y_1, x_2, y_2, state\rangle$, where $x_1, y_1, x_2$ and $y_2$ are used to represent the bounding box of the traffic light (i.e., a rectangular region around the traffic light within the image), and *state* represents the state of the traffic light (e.g., a stop state for a red traffic light). $x_1$ and $y_1$ are the x and y coordinate of the top left corner of the bounding box, and $x_2$ and $y_2$ are the x and y coordinate of the bottom right corner of the bounding box.

**Weather Transformations.** Autonomous vehicles drive in different weather environments, which may partially block traffic lights or reduce the visibility of traffic lights. For example, snowflakes or raindrops may happen to block traffic lights when the on-board camera captures the image. Therefore, weather environments may challenge the performance of traffic light detection. To simulate different weather environments, we design four transformations, i.e., rain (RN), snow (SW), fog (FG) and lens flare (LF). We use the image augmentation Python library `imgaug` [25] to realize RN, SW and FG. The basic idea is to generate and superimpose a layer of perturbation on each pixel of the image to mimic weather conditions. Specifically, for RN, we set the *drop_size* parameter to (0.1, 0.2) to control the size of raindrops, and set the *speed* parameter to (0.2, 0.3) to control the density of raindrops. For SW and FG, we set the *severity* parameter to 2 to control the concentration of the fog or snow. Besides, we use Adobe Photoshop to realize LF, which mimics the dazzling light effect captured by the camera. Specifically, we compose a new lens flare layer provided by Adobe Photoshop on the real-world traffic light image whose brightness and contrast are automatically adjusted [3]. Fig. 2(a-h) shows some sample traffic light images synthesized by our four weather transformations.

**Camera Transformations.** It is well known that the quality of the images taken by on-board cameras may affect the performance of traffic light detection models. Due to various models of on-board cameras and their various ages and degrees of damage, the image captured by cameras may not have a good quality (e.g., overexposure or underexposure). Moreover, autonomous vehicles may run at a high speed or turn at intersections, which might blur the captured images to some extend. To simulate such camera effects, we develop three transformations, i.e., overexposure (OE), underexposure (UE) and motion blur (MB). Similar to weather transformations, we use the library `imgaug` [25] to realize OE, UE and MB. For OE and UE, the image brightness is linearly adjusted by adjusting the lightness in the HSL color space, and we set the *severity* parameter to 4 and 1 to control the degree of exposure for simulating overexposure and underexposure respectively. For MB, a filter that uses a kernel is applied on an image through convolution, we set the kernel size parameter $k$ to 15 to control the degree of blur (i.e., a kernel size of $15 \times 15$ pixels is used). Fig. 2(i-n) illustrates some sample traffic light images synthesized by our three camera transformations.

**Traffic Light Transformations.** Traffic lights may have different states and be placed at different positions in real world. To enrich the diversity of positions and states of traffic lights in images, we design five traffic light transformations, i.e., changing color of traffic lights (CC), moving position of traffic lights (MP), adding traffic lights (AD), rotating traffic lights (RT) and scaling traffic lights (SC).

- CC is designed to change the color of traffic lights in an image. Specifically, we randomly select a subset of traffic lights $\hat{T} \subseteq T$, and locate each traffic light $t_i$ in $\hat{T}$ according to its bounding box and extract it (i.e., the region corresponding to the bounding box is blank after extraction). Then, we use content-aware patch provided by Adobe Photoshop to fill the blank region. Next, we use the HSV color space to change the hue of the extracted traffic light so that red becomes green and green becomes red, and also correspondingly change the traffic light state, i.e., $l_i.state$. Then, we flip the extracted traffic light upside down to ensure that the red light bulb is on the top or left of the traffic light. Finally, we use image fusion algorithm (e.g., Poisson blending) to paste the transformed traffic light back to the original region.
- MP aims to move the position of traffic lights in an image. Similar to CC, for each $t_i \in \hat{T} \subseteq T$, we extract $t_i$ and patch the blank region. Then, we paste $t_i$ to a new position by a horizontal offset $\delta$ (here we set $\delta$ to the width of the bounding box). Finally, we modify the bounding box coordinates in $l_i$ as follows.
$$l_i.x_1 \doteq l_i.x_1 + \delta,\ l_i.x_2 \doteq l_i.x_2 + \delta$$
- AD is designed to add traffic lights to an image. The number of newly added traffic lights is a random number between one and half of the original number of traffic lights. The basic idea is to copy original traffic lights and paste them to the image. In that sense, AD is similar to MP. The only difference is that here we do not patch the original traffic lights but keep them.
- RT aims to rotate traffic lights in an image so that horizontal traffic lights become vertical and vertical traffic lights become horizontal. Similar to CC, for each $t_i \in \hat{T} \subseteq T$, we extract $t_i$ and patch the blank region. Then, we determine the center of $t_i$, i.e., $(x_c, y_c) = (\frac{l_i.x_1+l_i.x_2}{2}, \frac{l_i.y_1+l_i.y_2}{2})$. In order to ensure that the red light bulb is on the top or left of the traffic light and the green light bulb is located at the bottom or right of the traffic light, we rotate $t_i$ by 90 degrees clockwise around the center if it is a horizontal traffic light, and we rotate it by 90 degrees counterclockwise if it is a vertical traffic light. Finally, we paste the transformed traffic light at the center of the original region, and correspondingly modify the bounding box coordinates in $l_i$ as follows.
$$l_i.x_1 \doteq x_c - \frac{l_i.y_2 - l_i.y_1}{2},\ l_i.y_1 \doteq y_c - \frac{l_i.x_2 - l_i.x_1}{2}$$
$$l_i.x_2 \doteq x_c + \frac{l_i.y_2 - l_i.y_1}{2},\ l_i.y_2 \doteq y_c + \frac{l_i.x_2 - l_i.x_1}{2}$$
- SC is designed to scale the size of traffic lights to mimic the effect of taking the image from a greater distance. We implement SC with the help of the batch processing function of Adobe Photoshop. First, we import each image into Adobe Photoshop and expand the canvas according to its original size (e.g., increase the width by 320 pixels and the height by 180 pixels for a 16:9 image). Then, we fill the expanded region with uni-color and patch it using the image inpainting technique in Adobe Photoshop. Finally, we scale the new image to its original size. Due to the scaling, the bounding box of each traffic light is scaled correspondingly as follows, where $w$ and $h$ denote the original width and height of the bounding box, and $(x_c, y_c)$ denotes the center.

$$l_i.x_1 \doteq x_c - (\frac{l_i.x_2 - l_i.x_1}{2} \times \frac{w}{w+320}),\ l_i.y_1 \doteq y_c - (\frac{l_i.y_2 - l_i.y_1}{2} \times \frac{h}{h+180})$$
$$l_i.x_2 \doteq x_c + (\frac{l_i.x_2 - l_i.x_1}{2} \times \frac{w}{w+320}),\ l_i.y_2 \doteq y_c + (\frac{l_i.y_2 - l_i.y_1}{2} \times \frac{h}{h+180})$$

Fig. 2(o-x) reports some sample traffic light images synthesized by our five traffic light transformations.

## 3 EVALUATION

We first introduce the research questions of evaluating TigAug, and then elaborate our evaluation setup, and finally present our results.

### 3.1 Research Questions

We design the following four research questions to evaluate the effectiveness and efficiency of TigAug.

- **RQ1 Robustness Evaluation**: How effective are our augmented traffic light images in identifying erroneous behaviors of existing traffic light detection models using our metamorphic relations?
- **RQ2 Retraining Evaluation**: How effective are our augmented traffic light images in improving the performance of existing traffic light detection models via retraining?
- **RQ3 Efficiency Evaluation**: How is the time overhead of TigAug with respect to synthesizing images and retraining models?
- **RQ4 Naturalness Evaluation**: How is the naturalness of augmented traffic light images, and does it affect the performance of existing traffic light detection models?

**RQ1** is designed to detect potential erroneous behaviors in existing traffic light detection models via investigating the performance difference between augmented testing data and real-world testing data against the models trained from real-world training data. **RQ2** aims to analyze the potential performance improvement after retraining the models with augmented training data resulted from twelve transformations. Specifically, we investigate the performance difference for both real-world testing data and augmented testing data. **RQ3** is designed to analyze the time cost of applying transformations as well as the time cost of retraining models. We aim to investigate the practical cost of TigAug's effectiveness in detecting erroneous behaviors and improving model performance. **RQ4** aims to assess the naturalness of augmented images by manually inspecting and excluding "unnatural" images in order to make the rest of them resemble real-world images. We further investigate how the cleaned augmented data affect the model performance.

### 3.2 Evaluation Setup

We run our evaluation with two traffic light datasets and four state-of-the-art traffic light detection models.

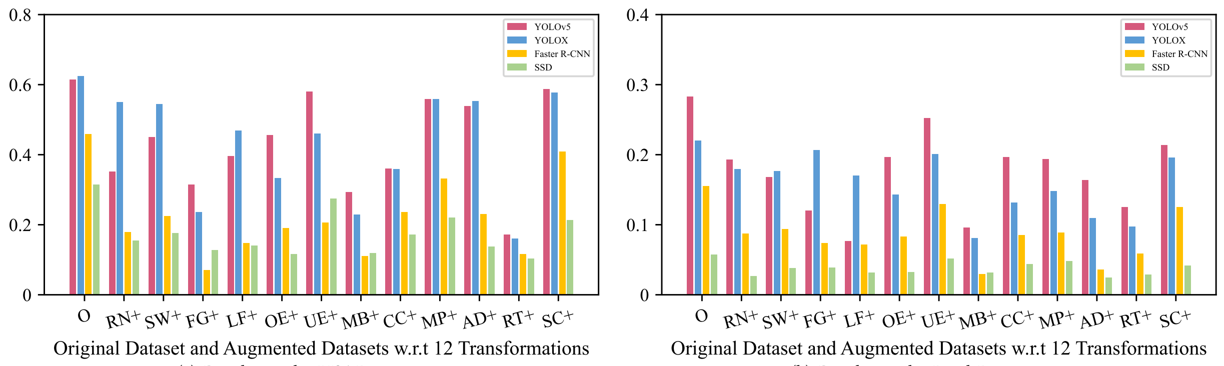
**Datasets.** We select two datasets, i.e., *LISA*[1] [22, 44] and *Bosch*[2] [8]. They are two of the most popular datasets in autonomous driving research area. *LISA* consists of continuous testing and training video sequences collected in California, USA. The sequences are captured by a stereo camera mounted on the roof of a vehicle driving under both night and daytime with varying light and weather conditions. The original *LISA* dataset totals 43,007 images and 113,888 annotated traffic lights. Meanwhile, the original *Bosch* dataset contains 13,427 images and about 24,000 annotated traffic lights. The annotations include bounding boxes of traffic lights as well as the current state of each traffic light. We filter monochrome images from the original *Bosch* dataset. We also remove duplicated images from both *LISA* and *Bosch* dataset. Moreover, we filter images with zero traffic light according to annotated data. Finally, we obtain preprocessed original datasets with a total number of 36,265 images and 109,475 annotated traffic lights in *LISA* as well as 10,300 images and 24,242 annotated traffic lights in *Bosch*. Each dataset is divided into training data, validation data and testing data by 4:1:1. We use `O` to represent these real-world datasets before augmentation.

To obtain augmented datasets for **RQ1**, **RQ2** and **RQ3**, we apply our twelve transformations on *LISA* and *Bosch*. As a result, we obtain 24 augmented datasets. We represent each augmented dataset using its corresponding transformation name and a '+' symbol; e.g., `RN+` represents the augmented dataset after applying our RN transformation. Besides, to support naturalness evaluation in **RQ4**, two of the authors manually inspect the "naturalness" of each transformed image in 12 augmented datasets resulted from *LISA*. To reduce the manual effort, we do not analyze the smaller dataset *Bosch*. The two authors first conduct a pilot labeling on randomly sampled 6,000 images to determine whether the transformed image is "natural". In the cases where there is a conflict between the two authors, a third author is involved to have a group discussion and reach agreements. The process takes 3 more rounds until the Cohen Kappa coefficient reaches 0.845. Finally, the two authors go through the remaining augmented data to filter "unnatural" images. The whole procedure takes 2 human-months. We represent each cleaned augmented dataset after manual analysis using its corresponding transformation name and a '-' symbol; e.g., `RN-` represents the cleaned augmented dataset after applying our RN transformation and manual cleaning.

**Models.** We choose four state-of-the-art traffic light detection models, i.e., *YOLOv5*[3], *YOLOX*[4], *Faster R-CNN*[5] and *SSD*[6]. These models are widely used in autonomous driving research [8, 28, 33, 40, 42]. Briefly, *YOLO* model and its generations (e.g., *YOLOv3*, *YOLOv5* and *YOLOv8*) [45] are cutting-edge and state-of-the-art models in a variety of tasks, including object detection, image segmentation and image classification. We choose *YOLOv5* because it is mature and stable. Further, *YOLOX* is a popular variant of *YOLO*, redesigned for anchor-free as well as other improvements for better performance. *Faster R-CNN* [46] is an object detection framework based on deep convolutional networks including a region proposal network and an object detection network. Both networks are trained for sharing convolutional layers for fast testing. *SSD* [35] is implemented based on a single shot multi-box detector.

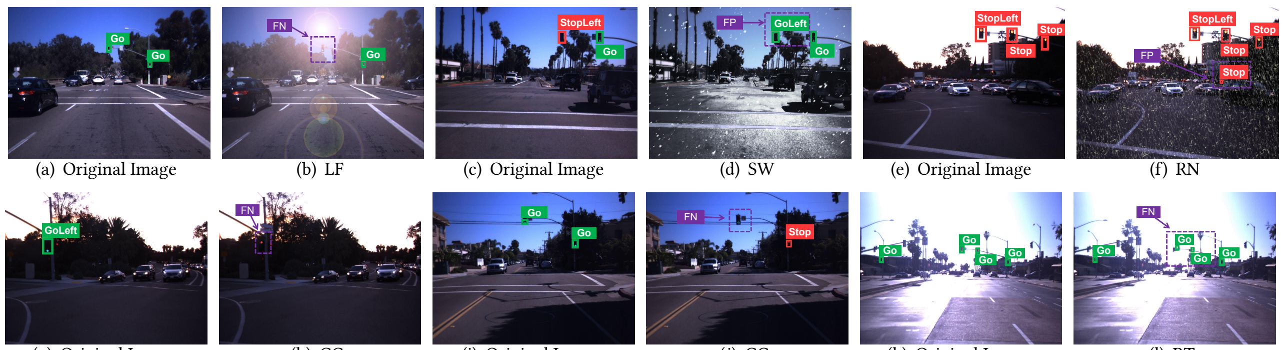
For **RQ1**, we obtain eight original models by training the four models *YOLOv5*, *YOLOX*, *Faster R-CNN* and *SSD* with the two original training datasets from *LISA* and *Bosch*. For **RQ2** and **RQ3**, we obtain 96 retrained models by retraining *YOLOv5*, *YOLOX*, *Faster R-CNN* and *SSD* with the 24 augmented training datasets. Specifically, to reduce the retraining time cost, we randomly choose 20% of each augmented training dataset to merge into the original training dataset. Notice that existing test case selection approaches [38] can be used here to further improve the retraining effectiveness. For **RQ4**, we obtain 24 retrained models by retraining *YOLOv5* and *YOLOX* with the 12 cleaned augmented training datasets from *LISA*. To align with the retraining in **RQ2** and **RQ3**, we first use the same 20% of augmented training dataset and then add extra cleaned augmented training samples if some of them are cleaned for "unnaturalness". Here we do not retrain *Faster R-CNN* and *SSD* because their training

[1]https://www.kaggle.com/datasets/mbornoe/LISA-traffic-light-dataset
[2]https://hci.iwr.uni-heidelberg.de/content/bosch-small-traffic-lights-dataset
[3]https://github.com/ultralytics/yolov5
[4]https://github.com/Megvii-BaseDetection/YOLOX
[5]https://github.com/ShaoqingRen/faster_rcnn
[6]https://github.com/amdegroot/ssd.pytorch



(a) Results on the *LISA* Dataset (b) Results on the *Bosch* Dataset

**Figure 3: mAP Comparison of the Original Models between Original and Augmented Testing Datasets**



(a) Original Image (b) LF (c) Original Image (d) SW (e) Original Image (f) RN

(g) Original Image (h) CC (i) Original Image (j) CC (k) Original Image (l) RT

**Figure 4: Samples of Traffic Light Images Revealing Erroneous Behaviors Found by TigAug**

time is much longer than *YOLOv5* and *YOLOX*, and their detection performance is much lower than *YOLOv5* and *YOLOX*.

We distinguish between original models and retrained models using the following notions. First, `M-O` represents an original model where `M` is a notion for model and `O` represents the original training dataset. Second, `M-`$\tau$`+` denotes a retrained model using augmented training dataset by a transformation $\tau \in \mathbb{W} \cup \mathbb{C} \cup \mathbb{L}$. For example, `M-RN+` denotes the retrained model using augmented training dataset by our *RN* transformation. Third, `M-`$\tau$`-` denotes a retrained model using cleaned augmented training dataset by a transformation $\tau \in \mathbb{W} \cup \mathbb{C} \cup \mathbb{L}$ after manual cleaning. For example, `M-RN-` denotes the retrained model using cleaned augmented training dataset by our *RN* transformation. Fourth, we also use concrete model's names to represent `M`. For example, `YOLOX-O` denotes the original *YOLOX* model using the original training dataset; and `YOLOX+` (resp. `YOLOX-`) denotes the retrained *YOLOX* model using (resp. cleaned) augmented training dataset without distinguish transformations.

**Metric.** We use mAP (a value between 0 and 1), as introduced in Sec. 2.1, to measure the detection performance of models.

**Environment.** We conduct the experiments on a Ubuntu 20.04.4 LTS server with 4 NVIDIA GeForce RTX 3090 GPUs, Intel Core i9-10980XE CPU with 3.00GHz processor and 128GB memory.

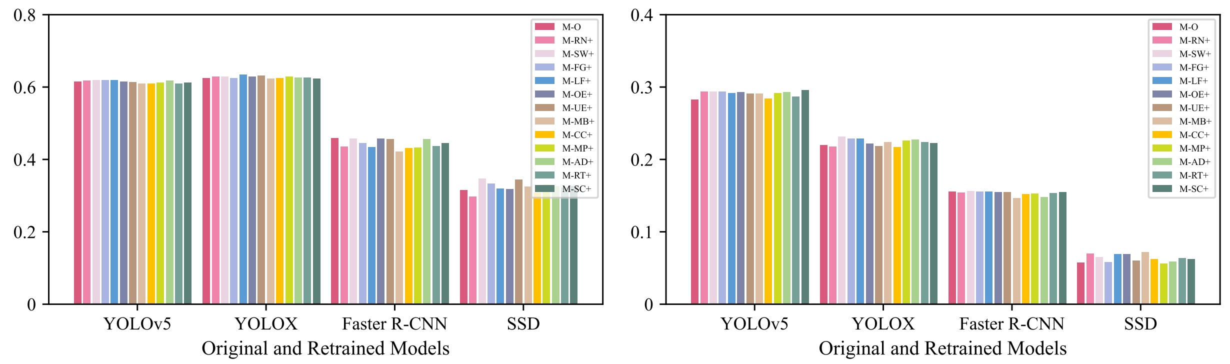
## 3.3 Robustness Evaluation (RQ1)

**mAP Drops of Original Models.** Fig. 3 reports the mAP of the original models on the original and augmented testing datasets. The mAP of all the four models on the augmented testing datasets decreases significantly across all the twelve transformations, when compared with that on the original testing datasets. Such mAP drops indicate that the original models are not robust to augmented images.

With respect to different models, *YOLOv5*, *YOLOX*, *Faster R-CNN* and *SSD* respectively suffer an average mAP drop of 31.4% (from 0.615 on the original testing dataset to 0.422 on the augmented testing dataset), 32.8% (from 0.624 to 0.419), 55.4% (from 0.459 to 0.205), and 48.3% (from 0.315 to 0.163) on *LISA* across all transformations. Similarly, they respectively suffer an average mAP drop of 41.3% (from 0.283 to 0.166), 30.5% (from 0.220 to 0.153), 48.4% (from 0.155 to 0.080), and 36.2% (from 0.057 to 0.036) on *Bosch*. Moreover, *YOLOv5*, *YOLOX*, *Faster R-CNN* and *SSD* respectively suffer a mAP drop between 0.031 (for UE on *Bosch*) and 0.443 (for RT on *LISA*), 0.013 (for FG on *Bosch*) and 0.464 (for RT on *LISA*), 0.026 (for UE on *Bosch*) and 0.388 (for FG on *LISA*), and 0.005 (for UE on *Bosch*) and 0.212 (for RT on *LISA*).

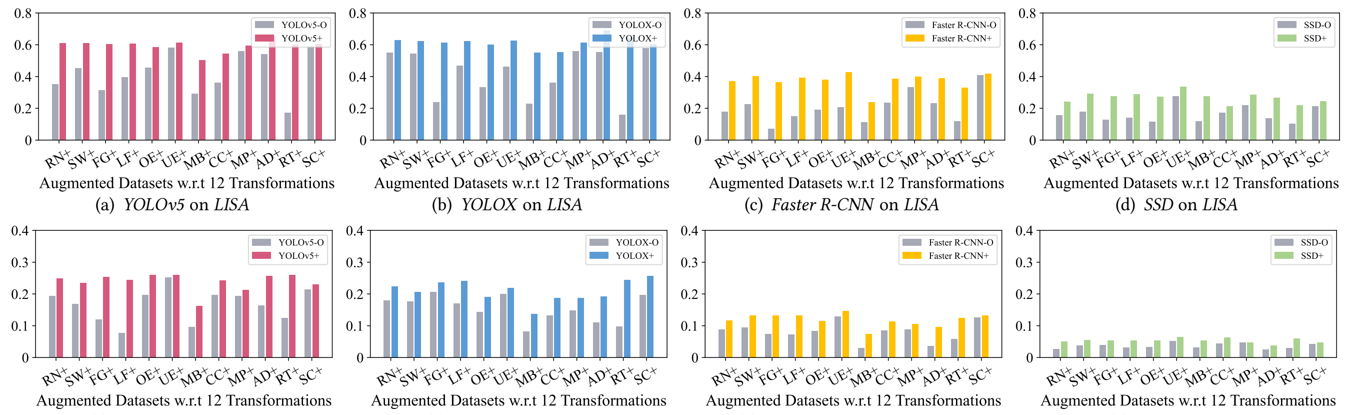
With respect to different transformations, the mAP decreases by 40.7%, 41.8% and 38.1% on average for weather, camera and traffic light transformations respectively. Specifically, the mAP decreases most significantly for FG (an average drop of 50.3%) among the four weather transformations, for MB (an average drop of 63.5%) among the three camera transformations, and for RT (an average drop of 63.9%) among the five traffic light transformations.

**Detected Erroneous Behaviors.** We further analyze the erroneous behaviors detected by TigAug that violate metamorphic relations, and summarize the false positives and false negatives reported by the original models on augmented testing datasets. Fig. 4 shows samples of traffic light images revealing erroneous behaviors. All transformations from the three families reveal erroneous behaviors. On one hand, regarding weather and camera transformations, there



(a) Results on the *LISA* Dataset (b) Results on the *Bosch* Dataset

**Figure 5: mAP Comparison between Original and Retrained Models on the Original Testing Datasets**



(a) *YOLOv5* on *LISA* (b) *YOLOX* on *LISA* (c) *Faster R-CNN* on *LISA* (d) *SSD* on *LISA*

(e) *YOLOv5* on *Bosch* (f) *YOLOX* on *Bosch* (g) *Faster R-CNN* on *Bosch* (h) *SSD* on *Bosch*

**Figure 6: mAP Comparison between Original and Retrained Models on the Augmented Testing Datasets**

are three kinds of erroneous behaviors. First, the original model fails to detect a traffic light, resulting in a false negative, as illustrated by Fig. 4(a) and 4(b). Second, the original model predicates a false label for a traffic light, resulting in a false positive, as illustrated by Fig. 4(c) and 4(d). Third, the original model recognizes a car's back light as a traffic light, resulting in a false positive, as shown by Fig. 4(e) and 4(f). On the other hand, regarding traffic light transformations, there are also three kinds of erroneous behaviors. First, the original model fails to detect a transformed traffic light, causing a false negative, as illustrated by Fig. 4(g) and 4(h). Second, the original model detects a transformed traffic light successfully, but fails to detect the unchanged traffic light or generates a wrong label, leading to a false negative or false positive, as shown by Fig. 4(i) and 4(j). Third, the original model detects a transformed traffic light, but finds a bounding box that is far too different from the ground truth, resulting in a false negative, as illustrated by Fig. 4(k) and 4(l).

***Summary.*** All the four models suffer a decrease in mAP on the augmented testing datasets across all the twelve transformations on the two datasets. On average, the mAP of each model decreases by 40.5%. Moreover, these models are more prone to erroneous behaviors for weather and camera transformations than for traffic light transformations. The average mAP drop ranges from 17.1% for SC transformation to 63.9% for RT transformation. Therefore, TigAug is effective in detecting erroneous behaviors for all models.

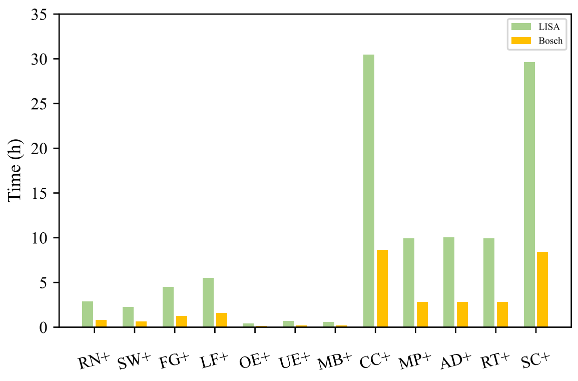
## 3.4 Retraining Evaluation (RQ2)

**mAP Changes on the Original Datasets.** Fig. 5 shows the mAP of the original and retrained models on the original testing datasets. Overall, the retrained models achieve a similar mAP as the original models on the original testing datasets.

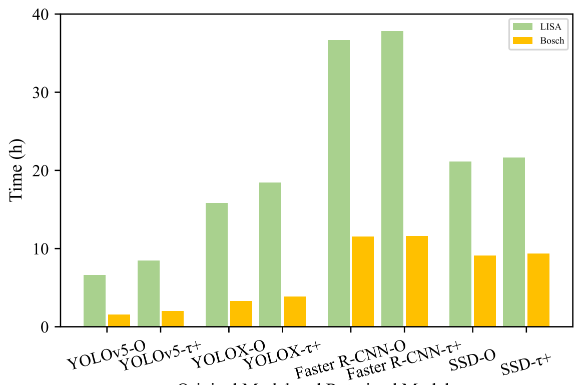
Specifically, with respect to retrained models, the average mAPs of retrained *YOLOv5*, *YOLOX* and *SSD* models across all transformations reach 0.615, 0.628 and 0.325 on the original *LISA* testing dataset (see Fig. 5(a)), which are slightly higher by 1.2% than those of the original models. Only the average mAP of retrained *Faster R-CNN* models become lower; i.e., the average mAP is 0.443, which is only 3.6% lower than the original model. Similarly, the average mAPs of retrained *YOLOv5*, *YOLOX* and *SSD* models are 0.292, 0.224 and 0.064 on the original *Bosch* testing dataset (see Fig. 5(b)), which are slightly higher by 5.6% than those of the original models. Only the average mAP of retrained *Faster R-CNN* models is lower; i.e., the average mAP is 0.153 (only 1.2% lower than the original model).

With respect to transformations, the average mAPs of retrained models slightly increase for SW, FG, LF, OE, UE, MP, AD and SC transformations, compared with the original models. SW transformation causes a largest increase of average mAP by 2.6%. The average mAPs of retrained models slightly decrease for RN, MB, CC and RT transformations, compared with the original models. CC transformation causes a largest decrease of average mAP by 1.5%.

**mAP Changes on the Augmented Datasets.** Fig. 6 reports the mAP of the original and retrained models on the augmented testing



(a) Time Cost of Transformations



(b) Time Cost of Model Retraining

**Figure 7: Time Overhead of TigAug**

datasets. Overall, all the retrained models have a significantly higher mAP than the original models on the augmented testing datasets.

Specifically, with respect to retrained models, the retrained *Faster R-CNN* models obtain the highest improvement in the average mAP among the four models, i.e., an average mAP improvement of 83.0% and 47.1% on the augmented *LISA* and *Bosch* testing datasets. Meanwhile, the retrained *YOLOv5*, *YOLOX* and *SSD* models respectively have an average mAP improvement of 40.2% and 43.3%, 46.2% and 37.0%, and 63.9% and 44.7% on the augmented *LISA* and *Bosch* testing datasets.

With respect to transformations, RT transformation contributes the most in improving the original models. It improves the average mAP by 210.5% and 118.7% on the augmented *LISA* and *Bosch* testing datasets, followed by FG transformation with an average mAP improvement of 193.5% and 59.6%. SC transformation contributes the least improvement, i.e., an average mAP improvement of 6.4% and 13.6% on the augmented *LISA* and *Bosch* testing datasets.

***Summary.*** Retrained models achieve similar mAP as original models on the original testing datasets, but have significantly higher mAP (i.e., 50.7% higher) than original models on the augmented testing datasets. RT transformation contributes the most in improving performance of original models. Therefore, TigAug is effective in improving model performance by feeding augmented data for model retraining.

## 3.5 Efficiency Evaluation (RQ3)

We first measure the time cost of transformations. The result is reported in Fig. 7(a). Generally, TigAug takes less than 10 hours for 22 of the 24 transformation tasks except for two, i.e., CC on *LISA* and SC on *LISA*, each of which takes around 30 hours. In total, TigAug consumes 137 hours to generate 24 augmented datasets. On average, TigAug takes 0.88 seconds to synthesize an image. Extremely, it takes a maximal of 3.03 seconds to synthesize an image by CC.

Then, we further analyze the time cost of model retraining. We measure each model's average retraining time across all transformations with respect to *LISA* and *Bosch* datasets. The result is reported in Fig. 7(b). Notice that we also report the training time of the original models for comparison. *YOLOv5* consumes the least retraining time of less than 10 hours, while *Faster R-CNN* consumes the most retraining time of more than 30 hours using the augmented *LISA* training datasets. This result also holds using the augmented *Bosch* training datasets. Besides, retraining takes an average of 12.35% more time than training the original models. In total, it takes 1,464 hours to train 8 original models using the original datasets and retrain 96 models using the augmented datasets. On average, TigAug takes 14 hours to retrain a model for improving performance.

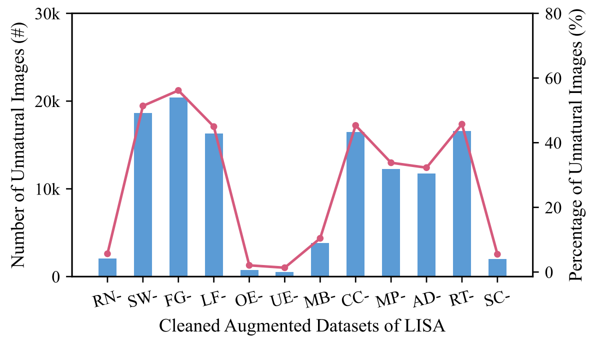
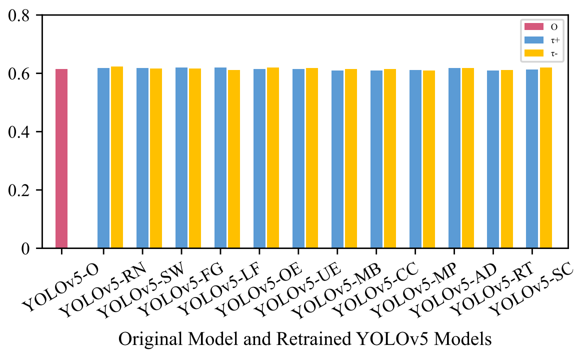


**Figure 8: Number and Percentage of Unnatural Images**

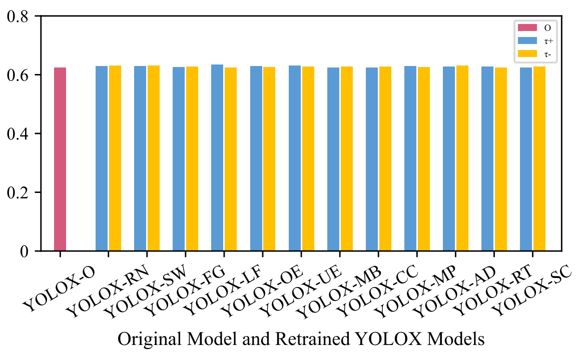
***Summary.*** On average, TigAug takes 0.88 seconds to synthesize an image, and takes 14 hours to retrain a model. Given the effectiveness in detecting erroneous behaviors and improving model performance, the time cost of TigAug is acceptable.

## 3.6 Quality Evaluation (RQ4)

**Number and Percentage of Unnatural Images.** We report the number and percentage of unnatural images (identified in our manual analysis) in each augmented dataset by a transformation in Fig. 8. Specifically, TigAug generates the least number of unnatural images by five transformations, i.e., RN, OE, UE, MB and SC. Less than 10% of the augmented images by these transformations are considered as unnatural. For the remaining transformations, the percentage of unnatural images ranges from 32% to 56%. On average, 27.9% of synthesized images by each transformation are considered as unnatural. Consequently, the number of cleaned augmented testing images is smaller than the number of original testing images by 1.3% to 56.2% due to the removal of unnatural images. Given that there are 6,166 original testing images, the cleaned augmented testing datasets are believed to be still sufficient.

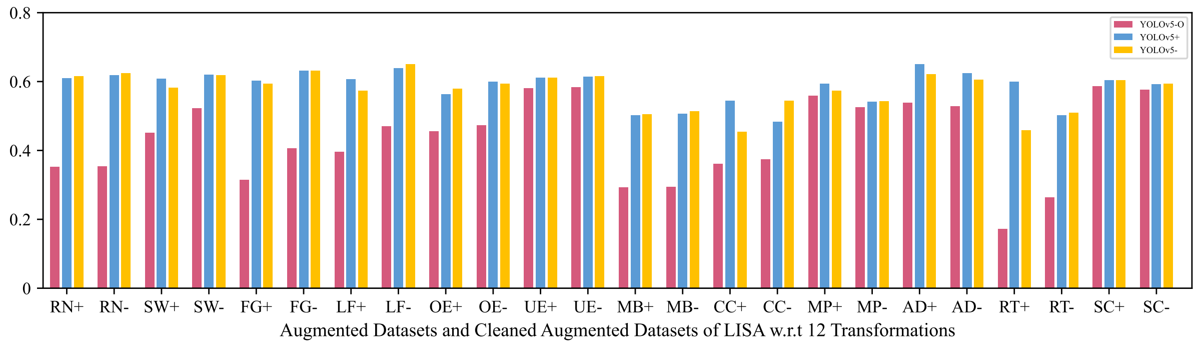


(a) *YOLOv5* Models

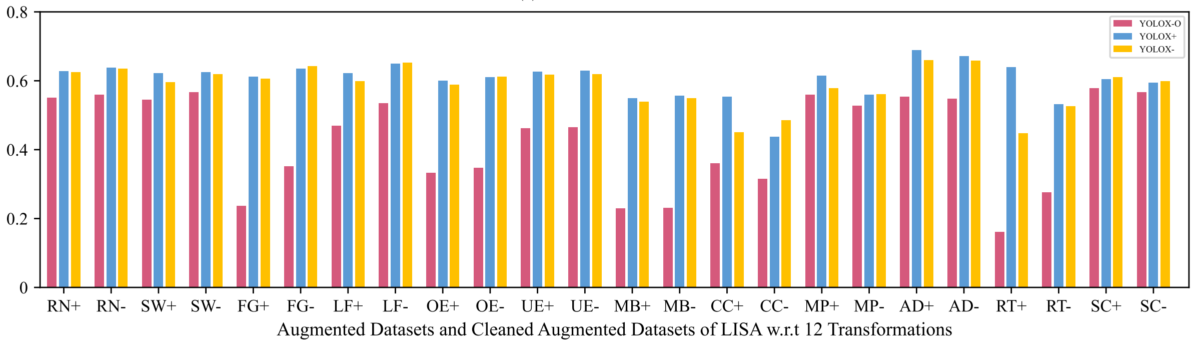


(b) *YOLOX* Models

**Figure 9: mAP of the Original and Retrained Models on the Original *LISA* Testing Dataset**



(a) *YOLOv5* Models



(b) *YOLOX* Models

**Figure 10: mAP of the Original and Retrained Models on the Augmented and Cleaned Augmented *LISA* Testing Dataset**

**Impact of Unnatural Images.** Fig. 9 shows the result of mAP comparisons among original models (i.e., the red bars), retrained models in **RQ2** without removing unnatural data (i.e., the blue bars), and retrained models without unnatural data (i.e., the yellow bars) on the original *LISA* testing dataset. Generally, these three kinds of models achieve similar mAP, indicating that those unnatural data will not affect models' detection capability on original real-world data.

Furthermore, similar to Fig. 9, Fig. 10 reports the result of mAP comparisons among these three kinds of models on the augmented *LISA* testing dataset (denoted with a '+' symbol, e.g., RN+) as well as on the cleaned augmented *LISA* testing dataset (denoted with a '-' symbol, e.g., RN-). First, the original models suffer a smaller mAP decrease after removing unnatural data. This is reasonable as the original models are not trained with such unnatural data. Second, on the augmented testing datasets, retrained models without removing unnatural data mostly have a slightly higher mAP than retrained models without unnatural data; i.e., the blue bars are mostly higher than the yellow bars. Hence, retrained models without removing unnatural data seems more robust. Third, on the cleaned augmented testing datasets, retrained models without removing unnatural data mostly have similar mAP as retrained models without unnatural data. This indicates that retraining models with those unnatural data will not affect models' capability on natural data.

***Summary.*** 27.9% of synthesized images by each transformation are considered as unnatural. However, these unnatural data, if used for retraining, will not affect models' capability on original real-world data and augmented natural data.

## 3.7 Threats to Validity

**Dataset Selection.** The quality and discrepancies of different driving environments in the datasets pose a threat to validity. To that end, we select two well-known datasets, i.e., *LISA* and *Bosch*, which are widely used in the research community on autonomous driving systems [8, 22, 44]. These datasets are collected in real-world environments with varying light and weather conditions, and consist of over 10k images and 20k annotated traffic lights. Therefore, both datasets cover a wide scope of driving environments. We believe our evaluation results could be generalized to other datasets.

**Model Selection.** The performance discrepancies of different traffic light detection models affect the validity. To that end, we select *YOLOv5*, *YOLOX*, *Faster R-CNN* and *SSD* to diversify the models and our evaluation results. They are widely used in the research community on autonomous driving systems [8, 28, 33, 40, 42]. We obtain consistent results from these four models, indicating that they could be generalized to other traffic light detection models.

**Transformations.** The parameter configuration in each transformation is empirically set to synthesize more natural images. Therefore, our evaluation results might not generalize to other parameter configurations. Besides, other transformations may exist in the three families. We plan to continuously enlarge our transformation set.

**Naturalness.** Traffic light detection models are fed with real-world images captured by cameras. The unnaturalness of synthesized images threats the validity. To that end, we design **RQ4** to manually exclude unnatural images and further investigate their impact. We find that the large part of unnatural images synthesized by SW and FG are those that make traffic lights invisible. Thus, even human beings cannot recognize the traffic lights, and it seems meaningless for models to detect. For LF, most of the synthesized unnatural images are caused by adding lens flare to night images. We plan to improve LF to not augment night images. For CC, MP, AD and RT, the unnaturalness of their synthesized images is caused by the oversized ground truth bounding box, which causes the fusion step to produce unnaturalness. Besides, we find that the unnatural images seem not affect models' detection capability on natural data. Therefore, we believe TigAug's augmentation capability is useful. It is interesting to investigate the correlation between unnatural data and bad data.

## 4 RELATED WORK

We review the closely related work in two areas: testing autonomous driving systems and metamorphic testing for deep learning.

### 4.1 Testing Autonomous Driving Systems

Testing has been recognized as one of the challenges in software engineering for autonomous driving systems [12, 29, 52]. Search-based testing has been widely investigated to find safety violations for autonomous driving systems [1, 2, 9, 17, 18, 21, 32, 50, 54, 60]. They often define a test scenario as a vector of multiple variables (e.g., vehicle speed, pedestrian position, and fog degree), and apply generic algorithms to search the input space for test scenarios that potentially violate a set of safety requirements (e.g., the distance between the vehicle and the pedestrian should be larger than a threshold). Differently, Tian et al. [55] generate safety-critical test scenarios by influential behavior patterns mined from real traffic trajectories.

Moreover, some advances have been made in testing DNN-based modules in autonomous driving systems. Pei et al. [43] design a neuron coverage-guided method, DeepXplore, to generate images of driving scenes by changing lighting conditions and occlusion with small rectangles. Tian et al. [56] also propose a neuron coverage-guided approach, DeepTest, to generate images of driving scenes more systematically through a more complete set of transformations. Different from these two approaches, Zhang et al. [59] propose to use generative adversarial network [19] to synthesize images of driving scenes with various weather conditions. To work with dynamically changing driving conditions, Zhou et al. [61] design DeepBillboard to generate adversarial perturbations that can be patched on billboards to mislead steering models. To test object detection models, Wang and Su [57] and Shao [49] propose to generate images of driving scenes by inserting extra object into background images. To test 3D object detection models, Guo et al. [20] apply affine transformation and weather transformation to augment LiDAR point clouds.

Besides, Gambi et al. [16] combine natural language processing with a domain-specific ontology to automatically generate car crash scenarios from police reports. Secci and Ceccarelli [47] summarize potential failure modes of a vehicle camera, and generate the corresponding failed images to analyze their effects on autonomous driving systems. Deng et al. [13] slice a driving recording into segments, reduce segment length by removing redundancy, and prioritize the resulting segments based on coverage of driving scene features. Despite these recent advances, many testing challenges still remain to better assure safety of autonomous driving systems [30, 37, 53].

To the best of our knowledge, except for Bai et al.'s work [7], none of the previous work is focused on the testing of traffic light detection models. While Bai et al. [7] generate traffic light images by changing the color of traffic lights, our work introduces a more complete set of transformations to augment traffic light images.

### 4.2 Metamorphic Testing for Deep Learning

As metamorphic testing [11, 48] can alleviate the test oracle problem, it has been widely used to test various deep learning systems apart from autonomous driving systems. For example, Dwarakanath et al. [14] use metamorphic testing to identify implementation bugs in image classifiers. Sun et al. [51] combine mutation testing with metamorphic testing to detect inconsistency bugs in machine translation systems. Chen et al. [10] design a property-based method to validate machine reading comprehension systems against seven metamorphic relations about different linguistic properties. Liu et al. [36] use transformation-specific metamorphic relations with Gini impurity guidance to test natural language understanding models of dialogue systems. Ji et al. [23] introduce three transformation families, i.e., characteristics mutation, noises injection, and reverberation simulation, based on metamorphic relations to test speech recognition systems. Yu et al. [58] propose a metamorphic testing approach to validate image captioning systems. Our work shares the same nature with these approaches by adopting metamorphic testing, but targets a different application domain of traffic light detection.

## 5 CONCLUSIONS

We have proposed and implemented a prototype tool, named TigAug, to automatically augment traffic light images for detecting erroneous behaviors and improving performance of traffic light detection models in ADSs. We conducted large-scale experiments to demonstrate the effectiveness and efficiency of TigAug. Our tool and data are available at https://zenodo.org/record/7645118/.

## REFERENCES

[1] Raja Ben Abdessalem, Shiva Nejati, Lionel C Briand, and Thomas Stifter. 2018. Testing vision-based control systems using learnable evolutionary algorithms. In *Proceedings of the 40th International Conference on Software Engineering*. 1016–1026.

[2] 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*. 143–154.

[3] Adobe. 2023. *How to add a lens flare in Adobe Photoshop*. Retrieved February 13, 2023 from https://www.adobe.com/creativecloud/photography/discover/lens-flare.html

[4] Anonymous. 2014. Incident 8: Uber Autonomous Cars Running Red Lights. In *AI Incident Database*, Sean McGregor (Ed.). Responsible AI Collaborative. Retrieved February 13, 2023 from https://incidentdatabase.ai/cite/8

[5] Daniel Atherton. 2022. Incident 434: Sudden Braking by Tesla Allegedly on Self-Driving Mode Caused Multi-Car Pileup in Tunnel. In *AI Incident Database*, Khoa Lam (Ed.). Responsible AI Collaborative. Retrieved February 13, 2023 from https://incidentdatabase.ai/cite/434/

[6] Claudine Badue, Rânik Guidolini, Raphael Vivacqua Carneiro, Pedro Azevedo, Vinicius B Cardoso, Avelino Forechi, Luan Jesus, Rodrigo Berriel, Thiago M Paixao, Filipe Mutz, et al. 2021. Self-driving cars: A survey. *Expert Systems with Applications* 165 (2021), 113816.

[7] Tongtong Bai, Yong Fan, Ya Pan, and Mingshuang Qing. 2021. Metamorphic Testing for Traffic Light Recognition in Autonomous Driving Systems. In *Proceedings of the IEEE 21st International Conference on Software Quality, Reliability and Security Companion*. 38–44.

[8] Karsten Behrendt, Libor Novak, and Rami Botros. 2017. A deep learning approach to traffic lights: Detection, tracking, and classification. In *Proceedings of the 2017 IEEE International Conference on Robotics and Automation*. 1370–1377.

[9] Raja Ben Abdessalem, Shiva Nejati, Lionel C Briand, and Thomas Stifter. 2016. Testing advanced driver assistance systems using multi-objective search and neural networks. In *Proceedings of the 31st IEEE/ACM international conference on automated software engineering*. 63–74.

[10] Songqiang Chen, Shuo Jin, and Xiaoyuan Xie. 2021. Validation on machine reading comprehension software without annotated labels: A property-based method. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 590–602.

[11] Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, Pak-Lok Poon, Dave Towey, TH Tse, and Zhi Quan Zhou. 2018. Metamorphic testing: A review of challenges and opportunities. *Comput. Surveys* 51, 1 (2018), 1–27.

[12] Krzysztof Czarnecki. 2019. Software Engineering for Automated Vehicles: Addressing the Needs of Cars That Run on Software and Data. In *Proceedings of the IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings*. 6–8.

[13] Yao Deng, Xi Zheng, Mengshi Zhang, Guannan Lou, and Tianyi Zhang. 2022. Scenario-based test reduction and prioritization for multi-module autonomous driving systems. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 82–93.

[14] Anurag Dwarakanath, Manish Ahuja, Samarth Sikand, Raghotham M Rao, RP Jagadeesh Chandra Bose, Neville Dubash, and Sanjay Podder. 2018. Identifying implementation bugs in machine learning based image classifiers using metamorphic testing. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 118–128.

[15] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. 2009. The pascal visual object classes (voc) challenge. *International journal of computer vision* 88 (2009), 303–308.

[16] Alessio Gambi, Tri Huynh, and Gordon Fraser. 2019. Generating effective test cases for self-driving cars from police reports. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 257–267.

[17] 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*. 318–328.

[18] Christoph Gladisch, Thomas Heinz, Christian Heinzemann, Jens Oehlerking, Anne von Vietinghoff, and Tim Pfitzer. 2019. Experience paper: Search-based testing in automated driving control applications. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering*. 26–37.

[19] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2020. Generative adversarial networks. *Commun. ACM* 63, 11 (2020), 139–144.

[20] An Guo, Yang Feng, and Zhenyu Chen. 2022. LiRTest: augmenting LiDAR point clouds for automated testing of autonomous driving systems. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*. 480–492.

[21] Jia Cheng Han and Zhi Quan Zhou. 2020. Metamorphic fuzz testing of autonomous vehicles. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*. 380–385.

[22] Morten Bornø Jensen, Mark Philip Philipsen, Andreas Møgelmose, Thomas Baltzer Moeslund, and Mohan Manubhai Trivedi. 2016. Vision for looking at traffic lights: Issues, survey, and perspectives. *IEEE Transactions on Intelligent Transportation Systems* 17, 7 (2016), 1800–1815.

[23] Pin Ji, Yang Feng, Jia Liu, Zhihong Zhao, and Zhenyu Chen. 2022. ASRTest: automated testing for deep-neural-network-driven speech recognition systems. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*. 189–201.

[24] Pengliang Ji, Ruan Li, Yunzhi Xue, Qian Dong, Limin Xiao, and Rui Xue. 2021. Perspective, survey and trends: Public driving datasets and toolsets for autonomous driving virtual test. In *Proceedings of the IEEE International Intelligent Transportation Systems Conference*. 264–269.

[25] Alexander B. Jung, Kentaro Wada, Jon Crall, Satoshi Tanaka, Jake Graving, Christoph Reinders, Sarthak Yadav, Joy Banerjee, Gábor Vecsei, Adam Kraft, Zheng Rui, Jirka Borovec, Christian Vallentin, Semen Zhydenko, Kilian Pfeiffer, Ben Cook, Ismael Fernández, François-Michel De Rainville, Chi-Hung Weng, Abner Ayala-Acevedo, Raphael Meudec, Matias Laporte, et al. 2020. *imgaug*. Retrieved February 13, 2023 from https://github.com/aleju/imgaug

[26] Prabhjot Kaur, Samira Taghavi, Zhaofeng Tian, and Weisong Shi. 2021. A survey on simulators for testing self-driving cars. In *Proceedings of the Fourth International Conference on Connected and Autonomous Driving*. 62–70.

[27] Srishti Khemka. 2021. Incident 347: Waymo Self-Driving Taxi Behaved Unexpectedly, Driving away from Support Crew. In *AI Incident Database*, Khoa Lam (Ed.). Responsible AI Collaborative. Retrieved February 13, 2023 from https://incidentdatabase.ai/cite/347/

[28] Jinkyu Kim, Hyunggi Cho, Myung Hwangbo, Jaehyung Choi, John Canny, and Youngwook Paul Kwon. 2018. Deep traffic light detection for self-driving cars from a large-scale dataset. In *Prceedings of the 21st International Conference on Intelligent Transportation Systems*. 280–285.

[29] Alessia Knauss, Jan Schroder, Christian Berger, and Henrik Eriksson. 2017. Software-Related Challenges of Testing Automated Vehicles. In *Proceedings of the IEEE/ACM 39th International Conference on Software Engineering Companion*. 328–330.

[30] Philip Koopman and Michael Wagner. 2016. Challenges in autonomous vehicle testing and validation. *SAE International Journal of Transportation Safety* 4, 1 (2016), 15–24.

[31] Khoa Lam. 2021. Incident 145: Tesla's Autopilot Misidentified the Moon as Yellow Stop Light. In *AI Incident Database*, Sean McGregor (Ed.). Responsible AI Collaborative. Retrieved February 13, 2023 from https://incidentdatabase.ai/cite/145

[32] Guanpeng Li, Yiran Li, Saurabh Jha, Timothy Tsai, Michael Sullivan, Siva Kumar Sastry Hari, Zbigniew Kalbarczyk, and Ravishankar Iyer. 2020. Av-fuzzer: Finding safety violations in autonomous driving systems. In *Proceedings of the IEEE 31st international symposium on software reliability engineering*. 25–36.

[33] Ziyue Li, Qinghua Zeng, Yuchao Liu, Jianye Liu, and Lin Li. 2021. An improved traffic lights recognition algorithm for autonomous driving in complex scenarios. *International Journal of Distributed Sensor Networks* 17, 5 (2021).

[34] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common objects in context. In *Proceedings of the 13th European Conference on Computer Vision*. 740–755.

[35] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. 2016. Ssd: Single shot multibox detector. In *Proceedings of the 14th European Conference on Computer Vision*. 21–37.

[36] Zixi Liu, Yang Feng, and Zhenyu Chen. 2021. DialTest: Automated testing for recurrent-neural-network-driven dialogue systems. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 115–126.

[37] Guannan Lou, Yao Deng, Xi Zheng, Mengshi Zhang, and Tianyi Zhang. 2022,. Testing of autonomous driving systems: where are we and where should we go?. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 31–43.

[38] Wei Ma, Mike Papadakis, Anestis Tsakmalis, Maxime Cordy, and Yves Le Traon. 2021. Test Selection for Deep Learning Systems. *ACM Transactions on Software Engineering and Methodology* 30, 2 (2021), 1–22.

[39] Sean McGregor. 2018. Incident 4: Uber AV Killed Pedestrian in Arizona. In *AI Incident Database*, Sean McGregor (Ed.). Responsible AI Collaborative. Retrieved February 13, 2023 from https://incidentdatabase.ai/cite/4

[40] Julian Müller and Klaus Dietmayer. 2018. Detecting traffic lights by single shot detection. In *Proceedings of the 2018 21st International Conference on Intelligent Transportation Systems*. 266–273.

[41] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. 2016. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on intelligent vehicles* 1, 1 (2016), 33–55.

[42] WeiGuo Pan, YingHao Chen, and Bo Liu. 2019. Traffic Light Detection for Self-Driving Vehicles Based on Deep Learning. In *Proceedings of the 15th International Conference on Computational Intelligence and Security*. 63–67.

[43] 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*. 1–18.

[44] Mark Philip Philipsen, Morten Bornø Jensen, Andreas Møgelmose, Thomas B Moeslund, and Mohan M Trivedi. 2015. Traffic light detection: A learning algorithm and evaluations on challenging dataset. In *Proceedings of the IEEE 18th international conference on intelligent transportation systems*. 2341–2345.

[45] Joseph Redmon and Ali Farhadi. 2017. YOLO9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 7263–7271.

[46] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Proceedings of the Advances in neural information processing systems*. 1–9.

[47] Francesco Secci and Andrea Ceccarelli. 2020. On failures of RGB cameras and their effects in autonomous driving applications. In *Proceedings of the IEEE 31st International Symposium on Software Reliability Engineering*. 13–24.

[48] Sergio Segura, Gordon Fraser, Ana B Sanchez, and Antonio Ruiz-Cortés. 2016. A survey on metamorphic testing. *IEEE Transactions on software engineering* 42, 9 (2016), 805–824.

[49] Jinyang Shao. 2021. Testing object detection for autonomous driving systems via 3d reconstruction. In *Proceedings of the IEEE/ACM 43rd International Conference on Software Engineering Companion*. 117–119.

[50] Yang Sun, Christopher M Poskitt, Jun Sun, Yuqi Chen, and Zijiang Yang. 2022. LawBreaker: An Approach for Specifying Traffic Laws and Fuzzing Autonomous Vehicles. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–12.

[51] Zeyu Sun, Jie M Zhang, Mark Harman, Mike Papadakis, and Lu Zhang. 2020. Automatic testing and improvement of machine translation. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 974–985.

[52] Zaid Tahir and Rob Alexander. 2020. Coverage based testing for V&V and Safety Assurance of Self-driving Autonomous Vehicles: A Systematic Literature Review. In *Proceedings of the IEEE International Conference On Artificial Intelligence Testing*. 23–30.

[53] Shuncheng Tang, Zhenya Zhang, Yi Zhang, Jixiang Zhou, Yan Guo, Shuang Liu, Shengjian Guo, Yan-Fu Li, Lei Ma, Yinxing Xue, et al. 2022,. A Survey on Automated Driving System Testing: Landscapes and Trends. *arXiv preprint arXiv:2206.05961* (2022,).

[54] Haoxiang Tian, Yan Jiang, Guoquan Wu, Jiren Yan, Jun Wei, Wei Chen, Shuo Li, and Dan Ye. 2022. MOSAT: finding safety violations of autonomous driving systems using multi-objective genetic algorithm. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 94–106.

[55] Haoxiang Tian, Guoquan Wu, Jiren Yan, Yan Jiang, Jun Wei, Wei Chen, Shuo Li, and Dan Ye. 2022. Generating Critical Test Scenarios for Autonomous Driving Systems via Influential Behavior Patterns. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–12.

[56] 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*. 303–314.

[57] Shuai Wang and Zhendong Su. 2020. Metamorphic object insertion for testing object detection systems. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. 1053–1065.

[58] Boxi Yu, Zhiqing Zhong, Xinran Qin, Jiayi Yao, Yuancheng Wang, and Pinjia He. 2022. Automated testing of image captioning systems. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*. 467–479.

[59] 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*. 132–142.

[60] Ziyuan Zhong, Zhisheng Hu, Shengjian Guo, Xinyang Zhang, Zhenyu Zhong, and Baishakhi Ray. 2022. Detecting multi-sensor fusion errors in advanced driver-assistance systems. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*. 493–505.

[61] Husheng Zhou, Wei Li, Zelun Kong, Junfeng Guo, Yuqun Zhang, Bei Yu, Lingming Zhang, and Cong Liu. 2020. Deepbillboard: Systematic physical-world testing of autonomous driving systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 347–358.