PerturbationDrive: A Framework for Perturbation-Based Testing of ADAS

Hannes Leonhard

Technical University of Munich, Germany

Stefano Carlo Lambertenghi

Technical University of Munich & fortiss, Germany

Andrea Stocco

Technical University of Munich & fortiss, Germany

Abstract

Advanced driver assistance systems (ADAS) often rely on deep neural networks to interpret driving images and support vehicle control. Although reliable under nominal conditions, these systems remain vulnerable to input variations and out-of-distribution data, which can lead to unsafe behavior.

We present PerturbationDrive, a testing framework to perform robustness and generalization testing of ADAS. The framework features more than 30 image perturbations from the literature that mimic changes in weather, lighting, or sensor quality and extends them with dynamic and attention-based variants. PerturbationDrive supports both offline evaluation on static datasets and online closed-loop testing in different simulators. Additionally, the framework integrates with procedural road generation and search-based testing, enabling systematic exploration of diverse road topologies combined with image perturbations. Together, these features allow PerturbationDrive to evaluate robustness and generalization capabilities of ADAS across varying scenarios, making it a reproducible and extensible framework for systematic system-level testing.

Keywords: ADAS testing, Autonomous Driving, Image perturbations, Search-based Testing, DNN testing.

Metadata

Table 1: Code metadata

Code metadata description	**Please fill in this column**
Current code version	v1.0.0
Permanent link to code/repository used for this code version	`https://github.com/ast-fortiss-tum/perturbation-drive.git`
Legal Code License	MIT
Code versioning system used	git
Software code languages, tools, and services used	Python 3.9, Unity (C#), pygame
Compilation requirements, operating environments and dependencies	Unix (x86/arm)
Link to developer documentation/manual	`https://github.com/ast-fortiss-tum/perturbation-drive/blob/Replication/README.md`
Support email for questions	lambertenghi@fortiss.org

1. Introduction

Advanced driver assistance systems (ADAS) use perception modules to interpret driving environments in real time for tasks such as object detection, segmentation, and control regression [1, 2, 3, 4, 5]. Deep neural networks (DNNs) represent the standard methodology for ADAS perception and currently deliver the best reported performance. Although accurate under nominal conditions, DNNs often fail to generalize to unseen domains. Exhaustive data collection is infeasible, and small shifts in lighting, weather, or viewpoint can cause perception errors [3, 6, 7] that can produce unsafe driving behaviors.

Model-level testing provides insights but ignores the closed-loop nature of driving, where perception continuously affects control [8, 9]. Large-scale validation requires simulators, since real-world testing is unsafe and would demand million of miles [10]. Platforms such as CARLA [11], Udacity [12], DonkeyCar [13], and NVIDIA DriveSim [14] support reproducible evaluation, but realistic adverse conditions often require custom assets. Perturbation-based methods address this gap by directly manipulating images, ensuring portability across datasets and simulators.

In our previous work, we presented PerturbationDrive [15], a library that integrates several perturbations from the literature for vision-based ADAS testing [16, 17, 18, 19, 20, 21, 22]. In this work, we extended PerturbationDrive in several directions. First, we implemented dynamic [23] and attention-based variants [24, 25] of the original perturbations. Additionally, we integrate perturbations into search-based testing for combined scenario–perturbation exploration [26, 27, 28, 29] and we added support to the CARLA simulator. PerturbationDrive supports both offline (component-level) [30, 31, 32] and online (system-level) evaluation [9, 8, 33, 34, 35] to enable systematic and reproducible evaluation of ADAS, covering both robustness and generalization across diverse driving scenarios.

2. The PerturbationDrive Framework

2.1. Objectives

The goal of PerturbationDrive is to provide a systematic framework for evaluating ADAS under controlled image perturbations and procedurally generated road scenarios. It consolidates perturbation techniques into a library with standardized configuration for type, intensity, and random seed, ensuring reproducibility and comparability across ADAS models and simulation environments. Beyond offline testing, PerturbationDrive supports closed-loop evaluation in simulators and integrates perturbations with procedural road generation, enabling both robustness benchmarking and generalization analysis in unseen conditions.

2.2. System Architecture

The framework comprises three components (Figure 1): the *Perturbation Controller*, the *Simulator Adapter*, and the *Benchmarking Controller*.

2.2.1. Perturbation Controller

The Perturbation Controller implements a library of image perturbations available from the literature [16, 17, 18, 19, 20, 21, 22]. Perturbations are grouped into three categories:

i) Static perturbations: frame-level modifications such as noise, blur, defocus, weather overlays, geometric distortions, affine transformations, graphic

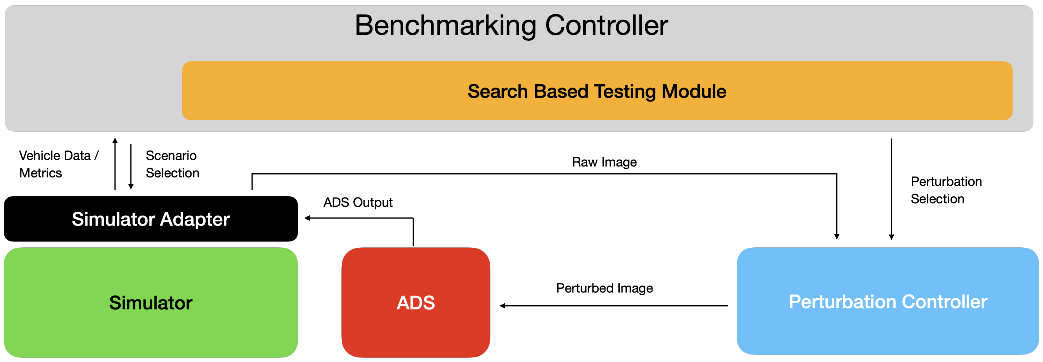


Figure 1: Overview of the PerturbationDrive framework.

patterns, and color or tone adjustments.
ii) Dynamic perturbations: temporal overlays (e.g., rain, snow, smoke, glare) that preserve consistency across frames.
iii) Attention-based perturbations: targeted distortions applied to salient regions identified by GradCAM or similar methods [36, 37]. All perturbations inherit from a common base class that defines the transformation interface for consistency and extensibility.

2.2.2. Simulator Adapter

The Simulator Adapter bridges the perturbation library and simulators. It intercepts raw camera frames, applies perturbations, and forwards modified frames to the ADAS under test. Current implementations support Udacity [38], DonkeyCar [39], and CARLA [11]; additional platforms (e.g., BeamNG [40], NVIDIA DriveSim [14]) can be added by implementing the adapter interface. The adapter also integrates with procedural road generation to produce diverse topologies and enforces a per-frame processing budget to maintain real-time execution at 30 FPS.

2.2.3. Benchmarking Controller

The Benchmarking Controller manages offline and online experiments [8]. In offline mode, it perturbs datasets and compares model outputs with ground truth or reference predictions. In online mode, it supervises closed-loop simulations, injecting perturbations in real time, logging frames, control actions, and vehicle states, and detecting failures such as collisions or lane departures. All parameters and execution traces are recorded to ensure reproducibility and enable replay of failure-inducing cases.

2.3. APIs and Modularity

The user-facing API supports three modes:
i) Image-level: perturb a single image for visualization or debugging.
ii) Dataset-level: perturb entire datasets to benchmark classifiers, detectors, or segmentation models.
iii) Online: perturb live simulator streams for end-to-end system evaluation.

Users specify perturbation type and intensity in all modes. The framework records parameters and random seeds to ensure consistent application. Modularity is achieved by separating concerns: the Perturbation Controller defines transformations, the Simulator Adapter ensures simulator-agnostic integration, and the Benchmarking Controller handles logging and execution. This layered design facilitates extension with new perturbations or simulators without altering existing code.

2.4. Search-Based Testing Integration

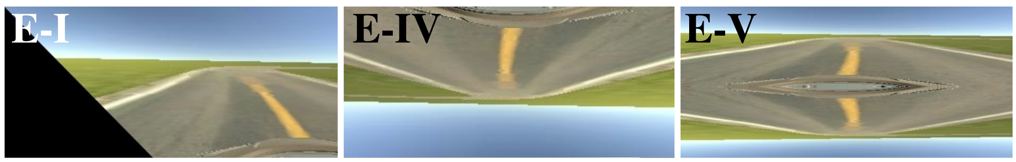
The combined space of perturbation type, intensity, and road scenario is too large for exhaustive evaluation. To address this, PerturbationDrive incorporates search-based software testing (SBST). Perturbations and scenarios are treated as search parameters, and candidate tests are generated using fitness functions such as input coverage, distance to failure, or diversity of cases. SBST integration prioritizes combinations most likely to expose system failures. Treating perturbations as first-class search variables enables unified exploration of environmental diversity and perceptual distortions, moving robustness evaluation beyond ad hoc perturbation studies toward systematic discovery of safety-critical failures in ADAS.

3. Implementation

3.1. Static Perturbations

PerturbationDrive provides 38 already implemented static perturbations. They are grouped into eight categories:
A) Noise perturbations, which mimic sensor or compression artifacts, including Gaussian, Poisson, impulse (salt-and-pepper), JPEG, and speckle noise.
B) Blur and focus perturbations, which reduce image sharpness through defocus, motion, zoom, Gaussian, or low-pass blur.
C) Weather perturbations, which simulate adverse conditions such as frosted glass, snow, fog, brightness shifts, and contrast changes.

(a) Valid.



(b) Invalid.

Figure 2: Examples of valid and invalid static perturbation types.

D) Distortion perturbations, which deform spatial structure via elastic deformation, pixelation, region blending, or sharpening.
E) Affine transformations, which alter global geometry through shear, scaling, translation, rotation, or reflection.
F) Graphic pattern perturbations, which overlay artificial structures such as splatter, dotted lines, zig-zags, edge maps, or cutout masks.
G) Color and tone adjustments, which change appearance by applying false colors, scrambling, histogram equalization, white balance, greyscale, saturation, or posterization.
H) Generative perturbations, which use deep models such as CycleGAN for domain remapping or style transfer for artistic overlays.

To ensure meaningful robustness evaluation, we manually inspected each perturbation and selected a standard set that preserves scene semantics and produces valid driving images (Figure 2a). Transformations that distort the scene unrealistically (e.g., removing the road, vertical flips) were excluded from the default configuration (Figure 2b).

3.1.1. Intensity Levels

Perturbation intensity was calibrated by visual inspection. Each perturbation was gradually increased until the scene was no longer reliably interpretable by a human observer. The maximum valid intensity was set just below this threshold, and the range was divided into five uniform levels.

(a) Overlay. (b) Particle.

Figure 3: Examples of overlay-based and Particle-based dynamic perturbation types.

3.2. Dynamic Perturbations

Dynamic perturbations in PerturbationDrive are implemented in two forms: video overlays and particle-based effects. These approaches maintain temporal consistency and remain simulator-agnostic since overlays are applied at the frame level rather than through engine-specific weather models.

3.2.1. Overlay-based effects

Each effect (e.g., rain streaks, snow, smoke, birds, glare) is represented by a pre-recorded green-screen video clip. During evaluation, chroma-key removal sets green pixels to transparent, blending only the perturbation elements into the scene. Temporal consistency is ensured using a `CircularBuffer`, which aligns simulator frames with the correct overlay frame, preserving natural motion such as continuous snowfall or rain streaks. Users may also supply custom overlays without coding: any green-screen video can be automatically processed and injected into the simulation.
An example spanning five seconds of simulation is shown in Figure 3a.

3.2.2. Particle-based effects

In addition to overlays, PerturbationDrive implements perturbations inspired by how raindrops or snowflakes interact with a physical camera lens. When precipitation strikes the lens, droplets or flakes attach to the glass, slowly accumulate, and then drift across the field of view under gravity and airflow. This creates localized occlusions that move over time, degrading visibility in a way that global weather overlays cannot reproduce. In PerturbationDrive, these effects are simulated by representing each droplet or flake as a particle with position, size, and velocity updated stochastically at every frame. Initial positions are sampled randomly (or from salient regions in the attention-guided variant), and their trajectories evolve according to random lateral drift, downward motion, and size adjustments. A `CircularBuffer` is used to maintain temporal consistency, ensuring that droplets and flakes persist and

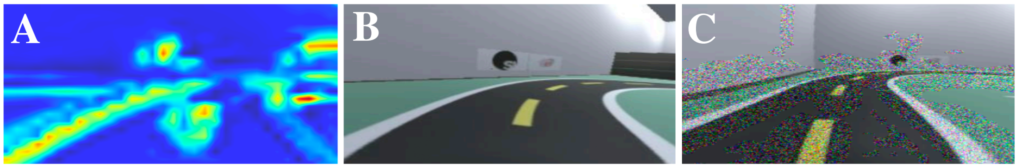


Figure 4: Example of attention-based perturbation.

move smoothly across frames instead of flickering. For rain, particles appear as semi-transparent streaks that may merge or slide, mimicking water on glass. For snow, flakes fall more slowly with wider drift, producing accumulation-like patterns. This particle-based design enables realistic simulation of precipitation on camera lenses, complementing static and overlay-based perturbations. An example spanning five seconds of simulation is shown in Figure 3b.

3.3. Attention-Based Perturbations

Attention-based perturbations in PerturbationDrive apply changes only to the parts of the image that the system under test considers important [36, 37].

The process begins with saliency extraction. By default, PerturbationDrive uses GradCAM to create a saliency map, which highlights the pixels the DNN relies on most. The map is resized to the input resolution and normalized to values between 0 and 1. Other attribution methods can be substituted as long as they return a pixel-level saliency map.

From this saliency map, a mask is generated. Pixels can be selected in several ways: keeping all values above a fixed threshold ϵ, selecting the top-n% of values (high-saliency regions such as lane markings or vehicles), or selecting the bottom-n% (low-saliency regions such as sky or trees). A random baseline can also be used, where the same number of pixels is chosen at random.

To ensure valid masks, small scattered areas are removed, and optional morphological closing is applied to form continuous regions. Soft masks with alpha blending can also be used, allowing gradual rather than sharp boundaries. In practice, for ADAS, background regions and ground outside the drivable surface are usually not relevant, so masks are focused on road areas and traffic participants.

Any static perturbation from the library (e.g., Gaussian noise, blur, occlusion, brightness change) can then be applied to the pixels within the mask. The final perturbed image I^* is computed as

$$I^* = M \odot \hat{I} + (1 - M) \odot I,$$

where I is the original image, $\hat{I}$ the perturbed image, and M the mask. This ensures that only masked areas are changed, while the rest of the image remains untouched. Saliency maps can be cached for offline evaluation or recomputed periodically during online testing. Masks may also be reused across short horizons to meet real-time constraints. An example of an attention-based perturbation is shown in Figure 4 (A: saliency map, B: original image, C: Perturbed image).

In addition to static masking, PerturbationDrive extends attention-based perturbations with dynamic precipitation effects guided by saliency. Prior work, such as AdvRain [41], has shown that placing synthetic raindrops at salient regions can create adversarial perturbations that mislead vision systems. Our approach differs in that the precipitation is not static: droplets are generated at salient regions on the lens and then drift until leaving the field of view, while snowflakes are emitted from salient regions and repeatedly cover important areas.

3.4. Perturbations in Search-Based Testing

Each test case is represented as a tuple (road scenario, perturbation type, intensity). Road scenarios are generated procedurally, while perturbations are applied to the camera stream.

A key requirement is an ordering of perturbations by effect strength; otherwise, type would be categorical, complicating evolutionary search. To address this, we established a ranking based on an empirical study across 2,450 scenarios (49 perturbations $\times$ 5 intensity levels $\times$ 10 roads) using the Udacity and DonkeyCar simulators, measuring degradation of an E2E lane-keeping and Advanced Cruise Control model, DAVE-2 [42]. Perturbations were sorted by the average failure rate (lane departures and speed changes) induced on the DAVE-2 model. This ranking allows perturbation type to be treated as an ordinal parameter in the SBST search space. The Benchmarking Controller integrates this ordering with the search-based testing framework OpenSBT [29]. During each iteration, candidate test cases are generated by combining road scenarios with perturbation parameters. Each case is executed in simulation, with outcomes (trajectories, collisions, lane departures) logged and converted into a fitness value. These fitness values guide the evolutionary search toward diverse and failure-inducing cases.

4. Usage

4.1. Installation

PerturbationDrive is available as a Python package. After cloning the repository and installing dependencies:

```
git clone https://github.com/ast-fortiss-tum/perturbation-drive.git
cd perturbation-drive
pip install -r requirements.txt
```

The library can be used by importing perturbation functions through the `ImagePerturbation` manager, or via the included simulator test scripts.

4.2. Offline Evaluation

Direct function calls. All perturbations are available as Python functions that take an image and an intensity parameter. For example:

```
from perturbationdrive import gaussian_noise, fog_filter
import cv2

image = cv2.imread("image.png", cv2.IMREAD_UNCHANGED)
perturbed = gaussian_noise(3, image.copy())   # intensity = 3
cv2.imwrite("gaussian.png", perturbed)
```

This approach is suitable for visualizing or debugging single perturbations.

A minimal example script is provided: `test_standalone_perturbations.py`.

Manager-based interface. Multiple perturbations can also be applied using the `ImagePerturbation` class, which dispatches calls by perturbation name:

```
from perturbationdrive import ImagePerturbation
import cv2

image = cv2.imread("0001.png", cv2.IMREAD_UNCHANGED)
perturbation_names = ["gaussian_noise", "fog_filter", "snow_filter"]

controller = ImagePerturbation(funcs=perturbation_names)

for p in perturbation_names:
    out = controller.perturbation(image.copy(), p, intensity=2)
    cv2.imwrite(f"0001_{p}.png", out)
```

A minimal example script is provided: `test_perturbation_manager.py`.

4.3. Online Evaluation

Closed-loop perturbations in simulators are demonstrated by two minimal scripts:

- `test_sim_udacity.py` for the Udacity simulator,
- `test_sim_donkey.py` for DonkeyCar.

Both scripts connect to the simulator, intercept camera frames, apply perturbations, and feed them to the model under test. The perturbation type and intensity are defined inside each script and can be modified by editing the corresponding calls to `ImagePerturbation`. A CARLA simulator adapter is provided in `examples/carla/`.

4.4. Extending the Library

Adding a perturbation. New perturbations can be added by defining a function in `perturbationdrive/` that follows the interface `func(intensity, image)`. They can also be registered for use with `ImagePerturbation`.

Adding a simulator. Additional simulators can be integrated by following the structure of `test_sim_udacity.py` and `test_sim_donkey.py`, where perturbations are injected into the frame-processing loop.

5. Expected Impact and Significance

PerturbationDrive provides a complete experimental environment for assessing the reliability of ADAS and the interplay between robustness and generalizability under controlled conditions. It supports systematic comparisons of image perturbations and road topology across diverse ADAS and simulation environments.

Compared to prior testing frameworks, PerturbationDrive offers the most comprehensive integration of evolutionary test generation with image perturbation. Its automated workflows minimize manual coding effort while ensuring reproducibility and extensibility. We hope that the tool will serve as an accelerator for researchers, students, and practitioners, enabling them to conduct reproducible robustness evaluations through intuitive interfaces that lowers entry barriers and facilitates both experimentation and analysis.

6. Conclusions and Future Work

We presented PerturbationDrive, a framework that combines image perturbations with evolutionary algorithms to systematically assess the quality of DNN-based ADAS. By integrating existing perturbations within a reusable framework, the tool enables automated and reproducible testing across multiple simulators. Future work will focus on enhancing the flexibility and extensibility of the framework, enabling seamless integration of custom ADAS, datasets, and generative perturbations, thus broadening its applicability for both research and practice.

7. Acknowledgments

This work was supported by the Bavarian Ministry of Economic Affairs, Regional Development, and Energy.

References

[1] S. Tang, Z. Zhang, Y. Zhang, J. Zhou, Y. Guo, S. Liu, S. Guo, Y.-F. Li, L. Ma, Y. Xue, Y. Liu, A Survey on Automated Driving System Testing: Landscapes and Trends, ACM Trans. Softw. Eng. Methodol. 32 (5) (Jul. 2023). `doi:10.1145/3579642`.

[2] E. Yurtsever, J. Lambert, A. Carballo, K. Takeda, A survey of autonomous driving: Common practices and emerging technologies, IEEE access 8 (2020) 58443–58469.

[3] Y. Li, L. Xu, Panoptic perception for autonomous driving: A survey (2024). `arXiv:2408.15388`.
URL `https://arxiv.org/abs/2408.15388`

[4] S. Grigorescu, B. Trasnea, T. Cocias, G. Macesanu, A survey of deep learning techniques for autonomous driving, Journal of Field Robotics 37 (3) (2020) 362–386.

[5] Y. Gao, M. Piccinini, Y. Zhang, D. Wang, K. Moller, R. Brusnicki, B. Zarrouki, A. Gambi, J. F. Totz, K. Storms, S. Peters, A. Stocco, B. Alrifaee, M. Pavone, J. Betz, Foundation models in autonomous driving: A survey on scenario generation and scenario analysis (2025). `arXiv:2506.11526`.
URL `https://arxiv.org/abs/2506.11526`

[6] S. Dodge, L. Karam, Understanding how image quality affects deep neural networks, in: 2016 Eighth International Conference on Quality of Multimedia Experience (QoMEX), 2016, pp. 1–6. doi:10.1109/QoMEX.2016.7498955.

[7] R. Geirhos, C. R. M. Temme, J. Rauber, H. H. Schütt, M. Bethge, F. A. Wichmann, Generalisation in humans and deep neural networks, in: Advances in Neural Information Processing Systems, Vol. 31, Curran Associates, Inc., 2018.

[8] A. Stocco, B. Pulfer, P. Tonella, Model vs system level testing of autonomous driving systems: a replication and extension study, Empirical Software Engineering 28 (3) (2023) 73. doi:10.1007/s10664-023-10306-x.

[9] F. U. Haq, D. Shin, S. Nejati, L. Briand, 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, 2020.

[10] BGR Media, LLC, 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 (2018).

[11] A. Dosovitskiy, G. Ros, F. Codevilla, A. López, V. Koltun, CARLA: an open urban driving simulator, CoRR abs/1711.03938 (2017).

[12] U. Team, Udacity's self-driving car simulator, https://github.com/tsigalko18/self-driving-car-sim (2019).

[13] Donkey Car, https://www.donkeycar.com/ (2021).

[14] NVIDIA Corporation, Nvidia drive sim - built on omniverse, https://developer.nvidia.com/drive/simulation, accessed: 2023-11-23 (2023).

[15] S. C. Lambertenghi, H. Leonhard, A. Stocco, Benchmarking image perturbations for testing automated driving assistance systems, in: Proceedings of the 18th IEEE International Conference on Software Testing, Verification and Validation, ICST '25, IEEE, 2025, p. 12 pages.

[16] D. Hendrycks, T. Dietterich, Benchmarking neural network robustness to common corruptions and perturbations, in: International Conference on Learning Representations (ICLR), 2019.

[17] D. Hendrycks, N. Mu, E. D. Cubuk, B. Zoph, J. Gilmer, B. Lakshminarayanan, AugMix: A simple data processing method to improve robustness and uncertainty, Proceedings of the International Conference on Learning Representations (ICLR) (2020).

[18] E. Rusak, L. Schott, R. S. Zimmermann, J. Bitterwolf, O. Bringmann, M. Bethge, W. Brendel, A simple way to make neural networks robust against diverse image corruptions, in: Computer Vision – ECCV 2020, Springer, 2020, pp. 53–69.

[19] J. Laermann, W. Samek, N. Strodthoff, Achieving generalizable robustness of deep neural networks by stability training, in: Pattern Recognition, Springer, 2019, pp. 360–373.

[20] A. Mikołajczyk, M. Grochowski, Data augmentation for improving deep learning in image classification problem, in: 2018 International Interdisciplinary PhD Workshop (IIPhDW), 2018, pp. 117–122. `doi:10.1109/IIPHDW.2018.8388338`.

[21] A. Stocco, B. Pulfer, P. Tonella, Mind the Gap! A Study on the Transferability of Virtual Versus Physical-World Testing of Autonomous Driving Systems, IEEE Transactions on Software Engineering 49 (04) (2023) 1928–1940. `doi:10.1109/TSE.2022.3202311`.

[22] J. Ayerdi, A. Iriarte, P. Valle, I. Roman, M. Illarramendi, A. Arrieta, Marmot: Metamorphic runtime monitoring of autonomous driving systems, ACM Trans. Softw. Eng. Methodol. 34 (1) (Dec. 2024). `doi:10.1145/3678171`.

[23] M. Daniali, E. Kim, Perception over time: Temporal dynamics for robust image understanding, in: 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2023, pp. 5656–5665. `doi:10.1109/CVPRW59228.2023.00599`.

[24] S. Munakata, C. Urban, H. Yokoyama, K. Yamamoto, K. Munakata, Verifying attention robustness of deep neural networks against semantic per-

turbations, in: 2022 29th Asia-Pacific Software Engineering Conference (APSEC), 2022, pp. 560–561. doi:10.1109/APSEC57359.2022.00080.

[25] S. Kitada, H. Iyatomi, Attention meets perturbations: Robust and interpretable attention with adversarial training, IEEE Access 9 (2020) 92974–92985.
URL https://api.semanticscholar.org/CorpusID:221949068

[26] M. H. Moghadam, M. Borg, S. J. Mousavirad, Deeper at the sbst 2021 tool competition: Adas testing using multi-objective search, in: 2021 IEEE/ACM 14th International Workshop on Search-Based Software Testing (SBST), 2021, pp. 40–41. doi:10.1109/SBST52555.2021.00018.

[27] D. Humeniuk, F. Khomh, G. Antoniol, Ambiegen: A search-based framework for autonomous systems testingimage 1, Science of Computer Programming 230 (2023) 102990. doi:https://doi.org/10.1016/j.scico.2023.102990.

[28] L. Sorokin, M. Biagiola, A. Stocco, Simulator ensembles for trustworthy autonomous driving testing (2025). arXiv:2503.08936.
URL https://arxiv.org/abs/2503.08936

[29] L. Sorokin, T. Munaro, D. Safin, B. H.-C. Liao, A. Molin, Opensbt: A modular framework for search-based testing of automated driving systems (2023). arXiv:2306.10296.

[30] K. Pei, Y. Cao, J. Yang, S. Jana, Deepxplore: Automated whitebox testing of deep learning systems, in: Proceedings of the 26th Symposium on Operating Systems Principles, SOSP '17, ACM, 2017, p. 1–18. doi:10.1145/3132747.3132785.

[31] Y. Tian, K. Pei, S. Jana, B. Ray, Deeptest: automated testing of deep-neural-network-driven autonomous cars, in: Proceedings of the 40th International Conference on Software Engineering, ICSE '18, ACM, 2018, p. 303–314. doi:10.1145/3180155.3180220.

[32] H. Zhou, W. Li, Y. Zhu, Y. Zhang, B. Yu, L. Zhang, C. Liu, Deepbillboard: Systematic physical-world testing of autonomous driving systems (2018). doi:10.48550/ARXIV.1812.10812.

[33] D. Liu, J. Zhao, A. Xi, X. H. Chao Wang, K. Lai, C. Liu, Data augmentation technology driven by image style transfer in self-driving car based on end-to-end learning, Computer Modeling in Engineering & Sciences 122 (2) (2020) 593–617.

[34] H.-J. Yoon, H. Jafarnejadsani, P. Voulgaris, Learning when to use adaptive adversarial image perturbations against autonomous vehicles, IEEE Robotics and Automation Letters 8 (7) (2023) 4179–4186. doi:10.1109/LRA.2023.3280813.

[35] R. Grewal, P. Tonella, A. Stocco, Predicting Safety Misbehaviours in Autonomous Driving Systems using Uncertainty Quantification, in: Proceedings of 17th IEEE International Conference on Software Testing, Verification and Validation, ICST '24, IEEE, 2024, p. 12 pages.

[36] X. Chen, M. Biagiola, V. Riccio, M. d'Amorim, A. Stocco, XMutant: XAI-based Fuzzing for Deep Learning Systems (2025). arXiv:2503.07222.
URL https://arxiv.org/abs/2503.07222

[37] A. Stocco, P. J. Nunes, M. d'Amorim, P. Tonella, ThirdEye: Attention Maps for Safe Autonomous Driving Systems, in: Proceedings of 37th IEEE/ACM International Conference on Automated Software Engineering, ASE '22, IEEE/ACM, 2022. doi:10.1145/3551349.3556968.

[38] Udacity, Udacity self-driving car simulator, https://github.com/udacity/self-driving-car-sim, accessed: [2024-01-15] (2021).

[39] M. E. Tawn Kramer, contributors, Donkeycar, https://www.donkeycar.com/ (2022).

[40] A. Gambi, P. Maul, M. Mueller, L. Stamatogiannakis, T. Fischer, S. Panichella, Soft-body simulation and procedural generation for the development and testing of cyber-physical systems, Tech. rep., BeamNG (2019).

[41] A. Guesmi, M. A. Hanif, M. Shafique, Advrain: Adversarial raindrops to attack camera-based smart vision systems, Information 14 (12) (2023). doi:10.3390/info14120634.
URL https://www.mdpi.com/2078-2489/14/12/634

[42] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, K. Zieba, End to end learning for self-driving cars., CoRR abs/1604.07316 (2016).