Run Run Shaw Library

香港城市大學
City University of Hong Kong

Copyright Warning

Use of this thesis/dissertation/project is for the purpose of private study or scholarly research only. ***Users must comply with the Copyright Ordinance.***

Anyone who consults this thesis/dissertation/project is understood to recognise that its copyright rests with its author and that no part of it may be reproduced without the author's prior written consent.

CITY UNIVERSITY OF HONG KONG
香港城市大學

Enhancing the Reliability of Intelligent Software Systems Through Novel Automated Testing Approaches
通過新型自動化測試方法增強智能軟件系統的可靠性研究

Submitted to
Department of Computer Science
電腦科學系
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
哲學博士學位

by

ZHANG Jingyu
張璟玉

AUGUST 2025
二零二五年八月

Declaration

I declare that the thesis entitled “Enhancing the Reliability of Intelligent Software Systems Through Novel Automated Testing Approaches” represents my own work, except where due acknowledgments are made, and that it has not been previously included in a thesis, dissertation, or report submitted to this university or any other institution for a degree, diploma, or other qualifications.

ZHANG Jingyu
AUGUST 2025

Abstract

The integration of deep learning (DL) techniques has catalyzed a paradigm shift in software development, moving systems from rule-based logic to data-driven intelligence. This is not merely an incremental improvement but a breakthrough that enables new classes of applications. For instance, DL-based models achieve outstanding performance in diverse domains, including computer vision, medical imaging, and financial fraud predictions. While the DL-based systems have exhibited remarkable success, they remain susceptible to out-of-distribution (OOD) input data, including adversarial samples, corrupted samples, and data with natural domain shifts. Understanding the challenges inherent in the usage of DL is crucial for developing and commercializing reliable and robust intelligent software systems. Typically, these challenges are mainly caused by the black-box characteristics of Deep Neural Networks (DNNs), resulting in difficulties in explaining and localizing the root cause of misbehaviors, thus making it harder to repair. One mainstream practice to alleviate these challenges is to generate representative and meanwhile, valid test cases (in the thesis, we use the terms test inputs and test cases interchangeably) that effectively reveal diverse faults in DNNs and use them to enhance the DNN under test.

In this thesis, we embark on a comprehensive exploration of four pivotal aspects in deep learning testing, each tackling crucial challenges in testing intelligent software systems: (1) Adversarial test input generation for autonomous driving systems (ADSs), (2) Transferable test input generation for ADSs, (3) Valid test input generation for image data, and (4) Cross-domain ensemble-based test input selection. We propose three novel DL-based approaches (UniAda, validity testing, and ensemble-based testing) and conduct empirical studies to address ongoing research challenges.

Specifically, recent research on testing intelligent systems has predominantly focused on generating adversarial test cases to trigger system misbehavior. In the realm of autonomous driving research, the focus of existing adversarial attack methods on end-to-end (E2E) ADSs has predominantly centered on misbehaviors of steering angle, which overlooks speed-related controls or imperceptible perturbations. To address

these challenges, we introduce UniAda, a multi-objective white-box attack technique with a core function that revolves around crafting an image-agnostic adversarial perturbation capable of simultaneously influencing both steering and speed controls. UniAda capitalizes on an intricately designed multiobjective optimization function with the adaptive weighting scheme (AWS), enabling the concurrent optimization of diverse objectives. Validated with both simulated and real-world driving data, UniAda outperforms five benchmarks across two metrics. This systematic approach establishes UniAda as a proven testing technique to enhance the reliability of DL-based E2E ADSs.

To ensure the generalizability of the generated adversarial test cases for ADSs, it is important to assess their transferability. A variety of transferability-enhancement approaches have been proposed, categorized into input transformation enhancements and attack objective enhancements. However, most of them are validated on image classification tasks only, there is limited research exploring methods to enhance attack transferability and assess the effectiveness of transferability-enhancement techniques on E2E ADSs. This part of the thesis seeks to fill the gap in autonomous driving research by conducting an empirical study on different transferability-enhancement techniques for steering models and identifying the most effective ones.

Numerous proposed automated test generation techniques are capable of automatically generating thousands of test cases. However, many of them lack a validity checking mechanism and have been shown to generate invalid inputs. Valid test input generation is pivotal in reliably enhancing the DNN under test, since the DNN is expected to only generalize within the valid input distribution, and invalid test inputs should not be predicted and may lead to unreliable testing. To mitigate this, we propose a novel framework for validity testing, which improves from existing testing frameworks by incorporating distribution awareness through joint optimization. By introducing distribution-based objectives, our proposed framework excels in generating valid test inputs, validated by both automated and human assessors, which establishes our method as an effective technique for valid test input generation.

Another challenge associated with a large number of unlabeled generated test cases lies in selecting a representative subset for manual labeling, which can then be used to enhance the performance of DNNs. Efficient test input selection and prioritization have emerged as promising approaches to ensure the quality of the selected test suite in supporting the improvement of intelligent DL systems. Identifying test inputs that are likely to reveal diverse DNN faults can significantly reduce manual labeling efforts and expedite the DNN repair process. However, when applying existing test input selection techniques in practice, we have observed three limitations: 1. Lack

of consideration for the consensus of results from multiple independent techniques (i.e., selecting inputs based on a fused score from multiple techniques); 2. Insufficient effectiveness evaluation of these techniques; 3. Limited scope of tasks and datasets. In this part of the thesis, we address these three limitations correspondingly by proposing an ensemble-based testing technique to consider the consensus decision, evaluating the effectiveness of these techniques with comprehensive criteria, and assessing the performance on both autonomous driving and malware detection domains. By implementing our proposed test selection technique, researchers can significantly improve the DNN repair process.

In summary, this thesis introduces novel approaches and empirical studies geared toward enhancing automated testing approaches for intelligent software systems. Through our proposed approaches and empirical insights, we aim to make meaningful contributions to the continual advancement of robust and effective techniques within these domains.

Keywords: Deep Learning Testing, Multi-objective Testing, Test Input Transferability, Test Input Validation, Ensemble-based Testing.

CITY UNIVERSITY OF HONG KONG
Qualifying Panel and Examination Panel

Surname:	ZHANG
First Name:	Jingyu
Degree:	Doctor of Philosophy
College/Department:	Department of Computer Science

The Qualifying Panel of the above student is composed of:

Supervisor(s)

Prof. KEUNG Wai Jacky	Department of Computer Science City University of Hong Kong

Qualifying Panel Member(s)

Prof. WANG Shiqi	Department of Computer Science City University of Hong Kong
Prof. SONG Linqi	Department of Computer Science City University of Hong Kong

This thesis has been examined and approved by the following examiners:

Prof. KEUNG Wai Jacky	Department of Computer Science City University of Hong Kong
Prof. WONG Hau San	Department of Computer Science City University of Hong Kong
Prof. LEUNG Wing Ho Howard	Department of Computer Science City University of Hong Kong
Dr. LAI Richard	Department of Computer Science and Information Technology La Trobe University

Acknowledgements

I would like to express my deepest gratitude to Professor Jacky Keung, my supervisor, for his invaluable mentorship and unwavering support throughout my Ph.D. journey. From the very beginning, he has provided not only technical guidance but also consistent encouragement that helped shape both my academic and professional development. His ability to recognize when I was struggling and to offer timely corrections and direction was critical in helping me stay on track. I am especially thankful for his meticulous supervision, structured management, and genuine care, which have been a source of strength and motivation during challenging times. I am also particularly grateful for his high standards and strict expectations, which continuously pushed me to go beyond my limits and strive for excellence. Beyond research, I deeply appreciate his support during my job hunting process, where his advice, encouragement, and belief in my potential have made a significant impact on my career path. Working under his supervision has been a privilege, and I am truly grateful for the opportunity to learn from him. My heartfelt thanks also go to Professor Lei Ma, who guided me during my visit to the University of Tokyo. His insightful advice on research directions and his constructive feedback greatly contributed to the improvement of my work. I would also like to thank Professor Wing Kwong Chan for his valuable guidance on my research project, which provided a strong foundation for my thesis.

Additionally, I am deeply appreciative of my Ph.D. qualifying panel members, Professor Shiqi Wang and Professor Linqi Song, for their time and thoughtful evaluation of my qualifying reports each year. Their comments and insights have been incredibly helpful to my research development. Furthermore, I am very grateful to my thesis examiners, Prof. Hau San Wong, Prof. Howard Leung, and Prof. Richard Lai, for their invaluable feedback and suggestions. Their constructive insights have greatly inspired me and contributed to improving the quality of my thesis.

I would like to extend my sincere thanks to my collaborators, Professor Yan Xiao and Dr. Xiaoxue Ma, for their steadfast guidance and ongoing support. Their

insightful feedback and generous assistance were especially invaluable during the more challenging phases of my research. I am truly grateful for their patience, encouragement, and the thoughtful advice they consistently provided throughout our collaborations.

I am also deeply thankful to my family: my mother, father, and cousin, for their constant encouragement and unwavering belief in me. Their understanding and support have been a source of strength during difficult times. A special note of gratitude goes to my husband, Mr. Zhenghan Yuan, for his love, support, and care over these past four years.

Finally, I would like to thank my dear lab members and friends, Yihan Liao, Yishu Li, and Xiaoxue Ma, for their companionship and support throughout this journey. Your friendship has made this experience truly meaningful.

Every setback lays the path toward future success,
Every sorrow paves the way for joy to come.

List of Publications

Published Journal Papers

1. **Jingyu Zhang**, Jacky Keung, Yan Xiao, Yihan Liao, Yishu Li, and Xiaoxue Ma. "Uniada: Universal Adaptive Multiobjective Adversarial Attack for End-to-End Autonomous Driving Systems." *IEEE Transactions on Reliability (TR)*, 2024, doi:10.1109/TR.2024.3394894.

2. Yihan Liao, **Jingyu Zhang***, Jacky Keung, Yan Xiao, and Yurou Dai, "Advancing Autonomous Driving System Testing: Demands, Challenges, and Future Directions," *Information and Software Technology*, 2025, doi: https://doi.org/10.1016/j.infsof.2025.107859.

3. Yihan Liao, Jacky Keung, **Jingyu Zhang***, Yurou Dai, and Shuo Liu. "Evaluate Inference Attacks: Attack and Defense against 2D Semantic Segmentation Models." *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 2025, doi: 10.1145/3732289.

4. Xiaoxue Ma, Jacky Keung, Xiao Yu, Huiqi Zou, **Jingyu Zhang**, and Yishu Li. "ATTSUM: A Deep Attention-Based Summarization Model for Bug Report Title Generation." *IEEE Transactions on Reliability (TR)*, 2022.

5. Yishu Li, Jacky Keung, Zhen Yang, Xiaoxue Ma, **Jingyu Zhang**, and Shuo Liu. "SimAC: Simulating Agile Collaboration to Generate Acceptance Criteria in User Story Elaboration." *Automated Software Engineering Journal (ASE)*, 2024.

6. Zhen Yang, Jacky Wai Keung, Xiao Yu, Yan Xiao, Zhi Jin, and **Jingyu Zhang**. "On the Significance of Category Prediction for Code-Comment Synchronization." *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2023.

Published Conference Papers

1. **Jingyu Zhang**, Jacky Keung, Xiaoxue Ma, Yihan Liao, Yishu Li, and Yicheng Sun, "Enhancing the Transferability of Adversarial Attacks for End-to-End Autonomous Driving Systems." *The 31st Asia-Pacific Software Engineering Conference, IEEE (APSEC 2024)*.

2. **Jingyu Zhang**, Jacky Keung, Xiaoxue Ma, Xiangyu Li, Yan Xiao, Yishu Li, and Wing Kwong Chan. "Enhancing Valid Test Input Generation with Distribution Awareness for Deep Neural Networks". *The 48th International Conference on Computers, Software, and Applications, IEEE (COMPSAC 2024)*.

3. **Jingyu Zhang**, Huitong Yang, Dai-Jie Wu, Jacky Keung, Xuesong Li, Xinge Zhu, and Yuexin Ma. "Cross-Modal and Cross-Domain Knowledge Transfer for Label-free 3D Segmentation." *The 6th Chinese Conference on Pattern Recognition and Computer Vision (PRCV 2023)*.

4. Yicheng Sun, Jacky Keung, **Jingyu Zhang***, Hi Kuen Yu, Wenqiang Luo, and Shuo Liu, "Unveiling Hidden Anomalies: Leveraging SMAC-LSTM for Enhanced Software Log Analysis". *The 48th Annual Computers, Software, and Applications Conference, IEEE (COMPSAC 2024)*.

5. Yicheng Sun, Jacky Keung, Hi Kuen Yu, Shuo Liu, Yihan Liao, and **Jingyu Zhang**. "Beyond Log Parsers: A Scalable AI-Driven Framework for Efficient Log Anomaly Detection in Software Engineering". *The 49th IEEE International Conference on Computers, Software, and Applications, IEEE (COMPSAC 2025)*.

6. Yishu Li, Jacky Keung, Xiaoxue Ma, Chun Yong Chong, **Jingyu Zhang**, and Yihan Liao. "LLM-based Class Diagram Derivation from User Stories with Chain-of-Thought Prompting." *The 48th IEEE International Conference on Computers, Software, and Applications, IEEE (COMPSAC 2024)*.

7. Yishu Li, Jacky Keung, Kwabena Ebo Bennin, Xiaoxue Ma, **Jingyu Zhang**, and Zhen Yang. "Agile Requirements Engineering in a Distributed Environment: Experiences from the Software Industry during Unprecedented Global Challenges." *The 48th IEEE International Conference on Computers, Software, and Applications, IEEE (COMPSAC 2024)*.

8. Yishu Li, Jacky Keung, Xiaoxue Ma, **Jingyu Zhang**, Zhen Yang, and Shuo, Liu. "Learning Gaps in Project-based Requirements Engineering Education - A Case Study of Student Projects." *The 9th International Symposium on Educational Technology, pp.239-243, IEEE (ISET 2023)*.

9. Yishu Li, Jacky Keung, Kwabena Ebo Bennin, Xiaoxue Ma, Yangyang Huang, and **Jingyu Zhang**. "Towards Requirements Engineering Activities for Machine Learning-enabled FinTech Applications" *The 30th Asia-Pacific Software Engineering Conference, IEEE (APSEC 2023)*.

Under Reviewed Journal Papers

1. **Jingyu Zhang**, Fan Wang, Jacky Keung, Xiaoxue Ma, Yuheng Huang, and Lei Ma. "Unified Ensemble-based Testing for Fault Detection and Retraining in Deep Neural Networks." *Empirical Software Engineering*, 2025. [Under Review]

Table of contents

Declaration ii

Abstract iii

Qualifying Panel and Examination Panel vi

Acknowledgements vii

List of Publications ix

List of figures xvi

List of tables xviii

1 Introduction 1

1.1 Test Case Generation for ADSs . . . 3
1.2 Test Case Transferability Analysis . . . 7
1.3 Test Case Validity Analysis . . . 8
1.4 Ensemble-based Test Selection . . . 10
1.5 Contributions of Thesis . . . 13
1.6 Organization of the Thesis . . . 16

2 Literature Review 18

2.1 Adversarial Attacks in DL-based ADSs . . . 18
2.1.1 DL in Autonomous Driving . . . 18
2.1.2 Adversarial Attacks in E2E ADSs . . . 19
2.2 Attack Transferability . . . 20
2.3 Test Case Validation . . . 21
2.3.1 Existing TIGs. . . . 21
2.3.2 Existing IVs. . . . 23

2.4 Test Case Selection 24
2.5 Ensemble Learning 25
2.6 Summary 26

3 UniAda: Universal Adaptive Multiobjective Adversarial Attack for End-to-End Autonomous Driving Systems 27
3.1 Preliminaries 28
3.2 Threat Model 30
3.3 Methodology 32
3.3.1 UniAda Overview 32
3.3.2 Universal Multi-objective Attack 33
3.3.3 Adaptive Weighting Scheme 34
3.3.4 Generating Adversarial Perturbations 36
3.4 Experimental Setup 39
3.4.1 Research Questions 39
3.4.2 Datasets & Models 39
3.4.3 Baselines 42
3.4.4 Evaluation Metrics 43
3.4.5 Hyperparameters 44
3.5 Results and Analyses 44
3.5.1 Visualization of Generated Adversarial Images 46
3.5.2 RQ1. UniAda Effectiveness. 46
3.5.3 RQ2. Adaptive Weighting Scheme Effectiveness. 50
3.5.4 RQ3. Multi-objective Attack Effectiveness. 54
3.6 Discussion 54
3.6.1 Transferability of Adversarial Attacks 54
3.6.2 Limitations 55
3.6.3 Threats to Validity 56
3.7 Chapter Summary 57

4 Enhancing the Transferability of Adversarial Attacks for End-to-End Autonomous Driving Systems 58
4.1 Preliminaries 59
4.1.1 Notations and Threat Model 59
4.1.2 Adversarial Attacks 60
4.1.3 Transferability-Enhancement Techniques 60
4.2 Study Design 63

4.2.1 Experimental Procedure . . . 63
4.2.2 Research Questions . . . 65
4.2.3 Experimental Setup . . . 66
4.3 Results and Analysis . . . 67
4.3.1 RQ1. Input Transformation Enhancement . . . 67
4.3.2 RQ2. Attack Objective Enhancement and its Comparison with Input Transformation . . . 70
4.3.3 RQ3. Runtime Efficiency Analysis . . . 73
4.3.4 Insights of Superior Performance . . . 74
4.4 Threats to Validity . . . 75
4.5 Chapter Summary . . . 76

5 Enhancing Valid Test Input Generation with Distribution Awareness for Deep Neural Networks 77
5.1 Empirical Study . . . 78
5.1.1 Empirical Setup . . . 78
5.1.2 Studied TIGs . . . 79
5.1.3 Studied IVs . . . 81
5.1.4 Studied Test Selection Metrics . . . 87
5.2 Our Approach . . . 89
5.3 Results . . . 92
5.3.1 Validity Assessments . . . 92
5.3.2 Improving DNNs with Enhanced Framework . . . 93
5.3.3 Practicability . . . 93
5.4 Threats to Validity . . . 94
5.4.1 Internal Validity . . . 94
5.4.2 External Validity . . . 94
5.5 Chapter Summary . . . 94

6 Unified Ensemble-Based Testing for Fault Detection and Retraining in Deep Neural Networks 96
6.1 Study Design . . . 97
6.1.1 Comprehensive Test Suite Evaluation . . . 98
6.1.2 Empirical Study . . . 101
6.1.3 Ensemble-based Testing Framework . . . 103
6.1.4 Study Overview . . . 105
6.2 Experimental Setup . . . 108

6.2.1 Studied Metrics 108
6.2.2 Datasets & Models 109
6.2.3 OOD Operators and selection budgets 112
6.2.4 Research Questions 113
6.2.5 Hyperparameters 114
6.3 Results 116
6.3.1 RQ1.1. Assessment of different clustering pipelines. 116
6.3.2 RQ1.2. Validation of cluster-to-fault correspondence. 119
6.3.3 RQ1.3 Comparison of different retraining strategies. 122
6.3.4 RQ2. Effectiveness of different metrics in two aspects. 124
6.3.5 RQ3. Effectiveness of our proposed testing framework. 127
6.4 Discussion 134
6.4.1 Threats to Validity 134
6.4.2 Limitations and Future Work 135
6.5 Chapter Summary 135

7 Conclusion and Future Work **137**
7.1 Thesis Contributions 137
7.2 Contributions to Software Engineering 139
7.3 Limitations and Future Research Work 140

References **143**

List of figures

1.1 Deep Learning Testing. 3
1.2 Overview of Modular ADS pipeline (top) and E2E Deep Learning-Based ADS pipeline (bottom) 4

2.1 Workflow of Input Validation: 1) Test Input Generation from seed inputs; 2) Test Input Validation by automated IVs. 22

3.1 Examples of adversarial attacks on image classification and regression tasks. Pixel values of perturbation τ are scaled for visibility. 28
3.2 UniAda Overview: Our attack technique contains two main optimization streams. The first one is the Universal Multi-objective Attack (in green) to generate a perturbation that is influential to multiple images and multiple car controls (i.e., steering S and acceleration A). The second one is the Adaptive Weighting Scheme (in yellow), which iteratively balances the objective weights to ensure similar training rates. Best viewed in color. 31
3.3 Adversarial images with different ϵ values. 44
3.4 Adversarial Images generated by UniAda: The first and second (third and fourth) rows are adversarial images generated from simulated (real-world) videos. Blue/Red arrows indicate the adversarial/original steering angle predicted from the image. The bottom right shows the acceleration error. 45
3.5 Traces of mean error (left) and AWS normalized weights (right) by UniAda attack in CILRS "Black Car". 53
3.6 Transferability of Adversarial attacks: Mean Error results in Steering (top) and Acceleration (bottom) of UniAda, DM, PA. Note that DM results for ME_A is not available. 55

4.1 The framework of adversarial attacks on steering models. Workflow 1: the Baseline Framework. Workflow 2: Input Transformation Enhancement. Workflow 3: Attack Objective Enhancement. m denotes the white-box model we craft the perturbation for. $m_2, ..., m_n$ denotes the black-box models being tested for transferability evaluation. 63
4.2 Attention Heatmap on Udacity sample image. 75

5.1 Invalid test inputs for MNIST and Udacity datasets. 81
5.2 The normalized confusion matrix depicting the average TP, FP, TN, FN values of all test suites for each dataset. 86

6.1 Overview of our study. Part 1: More comprehensive test suite evaluation. Part 2: Empirical study. Part 3: Our methodology. 107
6.2 Examples of test inputs from driving images (Udacity, Dave) and Android Packages (AndroZoo, Drebin). 111
6.3 Malware data: Heatmaps of high-quality clusters by the best pipeline (DeepDrebin, UMAP, DBSCAN). Visualizations of features of the misprediction set by BasicDNN and DeepDrebin, respectively. We also include noisy points as Cluster ID=-1 (top) for reference. 118
6.4 Malware data: Heatmaps of low-quality clusters by another pipeline (DeepDrebin, PCA, K-Means). Visualizations of features of the misprediction set by BasicDNN and DeepDrebin, respectively. 118
6.5 Driving data: Heatmaps of high-quality clusters by the best pipeline (ResNet-50, UMAP, DBSCAN). Visualizations of features of the misprediction set by Dave2V1 and Dave2V2, respectively. We also include noisy points as Cluster ID=-1 (top) for reference. 118
6.6 Driving data: Heatmaps of low-quality clusters by another pipeline (ResNet-50, PCA, K-Means). Visualizations of features of the misprediction set by Dave2V1 and Dave2V2, respectively. 119

List of tables

1.1 Contributions of the thesis 16

3.1 UniAda Hyperparameters: describe the meaning of each hyperparameter and the value used in the experiments. 37
3.2 Descriptions for the selected videos 40
3.3 Model Performance (MSE) on the provided validation set. 42
3.4 Mean Error Result Table: mean error results for 7 simulated videos of all six techniques, tested with CILRS and CILR models. The last line indicates the average result over 7 videos. Best results are highlighted in bold. We approximate the floating steering angle to degrees ° in ME_S and floating acceleration to km/h in ME_A for a more straightforward attack effect. - denotes unavailable results. . . . 47
3.5 Mean Error Result for 7 real-world driving videos of all six techniques with the MotionTransformer model under test. Avg. denotes the average results over 7 videos. 48
3.6 Mean Success Rate Result average over all testing videos for three ADSs: CILRS and CILR results are averaged over 7 Carla100 videos. MT results are averaged over 7 real-world driving videos. We tested thresholds δ_S=3.5, 14, 21, 28 in degrees and δ_A=4.6, 13.8, 23.0, 32.2 in km/h. 50
3.7 P-value results of Mean Error under CILRS, CILR and MT: results are shown in bold if they have a p-value $\leq$ 0.05, The positive/negative/equal signs indicate that UniAda has a better/worse/equal performance versus UniEqual. 51
3.8 P-value results of Success Rate under CILRS, CILR, and MT. 'NA' denotes unavailable results due to the same performance of UniAda and UniEqual. 51
3.9 Ablation study on CILRS and MT models, averaged over testing videos 54

4.1 MAE results (in degrees) of Input Transformation Enhancement (with L_{pred} as the attack objective). The superscript * indicates the white-box attack, and the others are black-box attacks (i.e., transferability). We highlight the results of the best technique in bold. 68
4.2 MAE results (in degrees) of Attack Objective Enhancement (with NO input transformation). The superscript * indicates the white-box attack, and the others are black-box attacks (i.e., transferability). We highlight the results of the best technique in bold. 71
4.3 MAE results for the Dave dataset. 72
4.4 The average runtime (in seconds) of perturbation generation and its standard deviation for Udacity. 73
4.5 The average runtime (in seconds) of perturbation generation and its standard deviation for Dave. 74

5.1 TIG Effectiveness: Input Validation results (%valid) with automated and human validators. For each IV, we highlight the best TIG in bold. 81
5.2 IV Effectiveness: IV vs. human labeling. For each TIG, we highlight the best IV's Acc in bold. AV denotes the average. Abbreviations for IVs: DV for DAIV, SD for DeepSVDD, KN for DeepKNN, and PN for PixelCNN++. 85
5.3 Results for Test Selection Metrics. Val. and inval. denote results with valid inputs and invalid inputs, respectively. Tot. and Avg. denote total and average results for the entire test suite. 90
5.4 Effectiveness of Joint Optimization (% valid). The positive/negative/equal signs indicate the improved techniques produce more/fewer/the same number of valid test inputs compared to the original techniques with the same seed inputs. 92
5.5 Improvement of DNNs Accuracy with "Random", "Original", and "Enhanced". 93

6.1 Description of studied datasets, models, tasks, performance, and corresponding OOD operators. 106
6.2 Description of studied OOD operators. 111
6.3 Summary of hyperparameters, including their descriptions, candidate ranges, selection methods, and the final selected values. 115

6.4 Silhouette and DBCV scores for each candidate pipeline, average over all datasets and models. Larger scores indicate better clustering quality. Best values are in bold. 117
6.5 Quantitative fault validation experiment on malware detectors (BasicDNN and DeepDrebin) with AndroZoo dataset and driving models (Dave2V1 and Dave2V2) with Dave dataset. In each row, we retrain the model under test on 85% of the samples in each cluster C_i. We report the accuracy of the retrained model on the remaining 15% of Cluster C_i (the third column) and the accuracy of the retrained model on all other clusters $C_{j\neq i}$ (the fourth column). 121
6.6 Average retraining performance (acc. improvements %) over all metrics and all OOD operators. The first column 'idr' indicates the ID ratio, idr=0% indicates ID+OOD=0%+100%. 123
6.7 Number of clusters across datasets and models, averaging over all budgets. 125
6.8 Accuracy improvements (%) of retraining (type I for malware data, type II for driving data) of the five selection metrics under 11 ID+OOD distributions, averaged over all settings (numbers shown in the parentheses). We highlight the best value in gray (for improvements) and yellow (for declines). 126
6.9 Fault detection results of our method, two variants (i.e., V1, V2), and five baselines (i.e., Rand, Ent, Gini, Dat, GD). Darker cells signify superior performance, while various colors denote the statistical significance among eight studied test selection metrics for each ID+OOD ratio (determined by the Scott-Knott ESD test with a p-value≤ 0.05). 128
6.10 Accuracy improvements of retraining of our method, two variants (i.e., V1, V2), and five baselines (i.e., Rand, Ent, Gini, Dat, GD). Cell color denotes the statistical significance by the Scott-Knott ESD test. 129

Chapter 1

Introduction

In the field of intelligent software testing, the generation of effective and generalizable test cases is pivotal for enhancing the reliability and robustness of intelligent software systems. Testing such systems mainly focuses on identifying the incorrectly learned properties and leads to data that can make the System Under Test (SUT) deviate from its expected behavior. Identifying and repairing misbehaviors are critical in the development and deployment of intelligent systems, which highlights the importance and necessity of comprehensive testing.

With the rapid rise of Deep Learning (DL) in safety-critical applications [1–3], at the early stage of research, several traditional software testing methodologies have been converted and applied to DL-based systems. Early efforts have introduced the concept of Neuron Coverage for measuring the parts of a DL system's logic exercised by a set of test inputs (we use the terms test inputs and test cases interchangeably throughout the thesis) based on the number of neurons activated (i.e., the output values are higher than a threshold by the inputs) [4]. This is improved based on the code coverage in traditional software testing and is aimed at testing how thoroughly a set of inputs tests the SUT. Following this line, many techniques (e.g., k-Multisection NC, Neuron Boundary Coverage, Top-K NC, Strong Neuron Activation Coverage) have been proposed to test the SUT with the combination of NC [5]. The line of DL testing research aims at efficiently generating tests and prioritizing tests from numerous unlabeled ones that can discover diverse faults as early as possible are also two important research directions. Fig. 1.1 illustrates the overview of DL testing. For test case generation, the aim is to generate misbehavior-inducing inputs for the SUT [6–8, 4]. There are various methods that can be leveraged to generate misbehavior-inducing synthetic test inputs, such as adversarial attacks [9, 10], and

transformation techniques (e.g., weather transformation, corruption) [11, 4]. Another way to collect misbehavior-inducing inputs is from natural out-of-distribution (OOD) data, including data collected from different sources or times [12]. However, most of the data are usually unlabeled (i.e., no human labels) in either collection method, and it takes a huge effort to label every collected sample. Facing numerous unlabeled test cases, it is important to effectively prioritize a small set of tests to be labeled and use them to enhance the SUT, mostly through model retraining. Test prioritization techniques (we use the terms test prioritization and test selection interchangeably throughout the thesis) are categorized into white-box and black-box methods, based on the information they use. For white-box techniques, we have access to full model information. Test selection techniques include uncertainty-based techniques that utilize the prediction uncertainty [13, 14] and surprise-based techniques that utilize the model surprise of the input [15]. For black-box techniques, no model information is available, so some researchers directly utilize raw data features only to select inputs. For example, Aghababaeyan *et al.* [16] selects the test suite with higher feature diversity, extracted by the VGG-16 [17]. After the selection and labeling of test inputs, they are used to enhance the SUT, normally through retraining with the labeled set [4, 18]. This is expected to enhance the feature that is insufficiently represented during the training process.

In this thesis, our attention lies on four aspects in DL testing: **(1) test case generation for autonomous driving systems** (adversarial white-box testing), **(2) test case transferability analysis** (transferable black-box testing), **(3) test case validity analysis** (input validity & realism), and **(4) ensemble-based test selection for DNNs** (test prioritization & evaluation strategy).

For aspects (1), (3), and (4), we proposed novel approaches to enhance the testing framework (e.g., test case generation and selection) for different SUTs, including Autonomous Driving Systems (ADSs), malware detectors, and image classifiers. For the second task (2), we conduct an empirical study that explores how existing transferability-enhancement techniques designed for the image classification task can be applied in the autonomous driving field. These approaches and empirical studies address the underlying research challenges comprehensively.

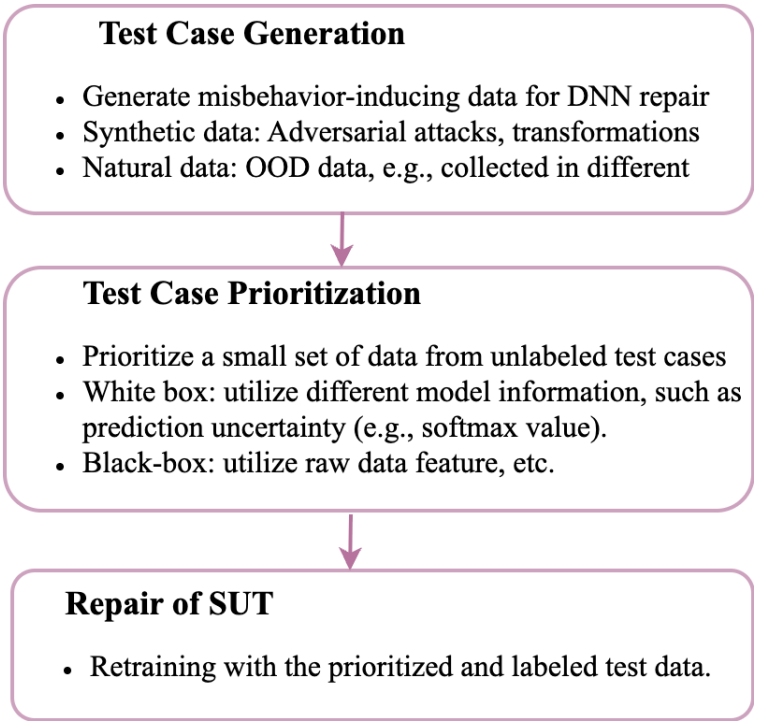


Fig. 1.1 Deep Learning Testing.

1.1 Test Case Generation for ADSs

Autonomous driving has brought about a transformative shift in the driving experience, showcasing the immense potential to significantly reduce accidents attributed to human errors and alleviate traffic congestion challenges [19].

There are two main approaches to implementing the ADS: modular [20] and End-to-End (E2E) [21]. Fig. 1.2 gives a simple architecture overview of these two systems. The conventional modular approach decomposes the system into interconnected modules encompassing perception, decision-making, planning, and control [22, 23, 20]. Recent advancements in deep learning have spurred interest in E2E methods, which combine multiple modules into a single deep neural network (usually CNN-based) model. E2E ADSs predominantly input sensor data (e.g., RGB images) and directly output numerical control actions such as steering and acceleration. While E2E ADSs have exhibited remarkable success [1, 24–26], they remain susceptible to malicious input data, including misleading corner cases or adversarial examples [27, 11, 28, 4].

Existing studies have delved into different methods to generate misbehavior-inducing test cases for the SUT (ADSs in our case), such as generating

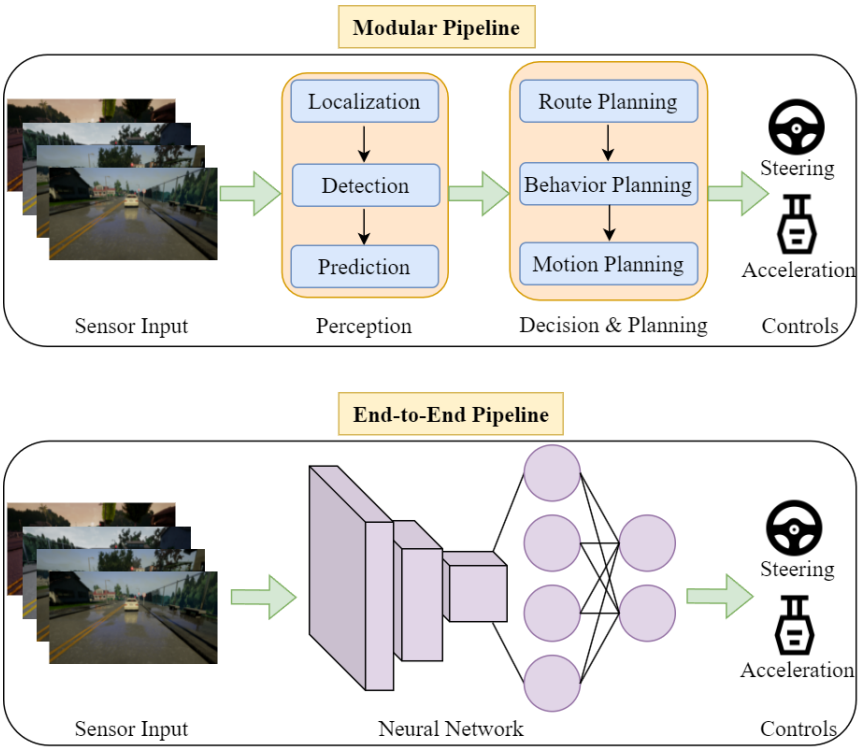


Fig. 1.2 Overview of Modular ADS pipeline (top) and E2E Deep Learning-Based ADS pipeline (bottom)

transformed or adversarial images as inputs [11, 27, 4, 28]. Usually, transformed images result from noticeable transformations applied to original inputs. Adversarial examples [29–33] involve subtle imperceptible perturbations to original data, avoiding human detection. These studies explore diverse transformations or perturbations to the input images and analyze their impact on steering actions. These range from modifying weather conditions [11, 34, 35] to introducing small black occlusions [4]. However, these efforts fall short in comprehensively testing ADSs, marked by three limitations.

- **Limited Vehicle Controls Testing**: Existing studies [27, 36, 28, 11] mainly concentrate on attacking steering control, while neglecting speed-related controls. This falls short of testing the security of real-world driving. Notably, no literature exists on targeting image inputs solely to trigger errors in both steering and acceleration controls for E2E ADSs based on deep learning.

- **Constrained Testing Scenarios**: Current testing scenarios lack diversity. For instance, SelfOracle proposed by Stocco *et al.* [37] exclusively focuses on highway scenarios, excluding intersections, traffic lights, pedestrians, and other vehicles. Similarly, attack techniques like DeepBillboard [27] and PhysGAN [28] concentrate on driving scenarios containing billboards. The state-of-the-art technique, DeepManeuver [36], exclusively performs attacks within a simulator, lacking the testing of real-world scenarios.

- **Image-specific or Unnatural Perturbations**: Many existing techniques generate perturbations or transformations specific to individual images [11, 4, 35, 34]. Alternatively, they produce perturbations easily detectable by humans, such as unnatural billboard replacements [27, 28, 36]. These limitations prompted our exploration of an adversarial perturbation—a subtle, human-imperceptible modification capable of consistently inducing errors in both speed and steering controls across a sequence of images (e.g., a driving record).

In addressing these shortcomings, we design UniAda that leverages adaptive multi-objective learning to generate universal adversarial perturbations for ADSs. Notably, we term an adversarial perturbation as **universal** when it consistently triggers errors larger than a predefined threshold across most images sampled from the data distribution [38, 39], thereby transcending image-specific constraints. There could still exist a limited number of images within the distribution where the generated universal perturbation fails to trigger errors surpassing the predefined threshold. In

this part of the thesis, we focus on the case where the distribution represents the set of images describing a driving scenario (e.g., car stops for a jaywalking pedestrian). To demonstrate the universality, we report the percentage of images that our generated perturbation attacks successfully by the Success Rate metric with four different tested thresholds for both objectives. The results have shown that UniAda can successfully attack 96.3% of images with steering error larger than 3.5°, which is a substantial improvement compared to baselines.

More specifically, UniAda optimizes a multi-objective function during perturbation generation. To achieve universality, UniAda establishes a joint optimization function through the summation of multi-objective functions associated with each image within the input driving record. Through iterative gradient descent, the joint function is minimized, facilitating the search for the perturbation. The resultant perturbation consistently triggers errors across multiple model predictions for most input images.

Instead of rudimentary uniform weighting or exhaustive grid search for objective weights, we propose the Adaptive Weighting Scheme (AWS), inspired by Chen *et al.* [40]. AWS, a gradient-based weight adjustment method, strives to balance varying objective weights to ensure all objectives are trained at a similar rate. This empowers UniAda to iteratively determine the optimal weight combination, enhancing the acquisition of shared feature representations for diverse objectives. To test the effectiveness of AWS, we propose a variant of UniAda, dubbed UniEqual, which shares the identical algorithm concepts but omits the use of AWS. We also conduct statistical analysis (i.e., t-test) to intuitively show the difference between UniAda and UniEqual, indicating that AWS brings a statistically significant improvement in most cases.

We evaluate UniAda on 14 urban driving videos selected from four datasets (i.e, Carla100 [25], Kitti [41], Udacity [26], Dave [42]) for two objectives (Steering and Acceleration) with three victim ADSs (i.e, CILRS, CILR, MotionTransformer). Our results demonstrate UniAda's superiority over five benchmarks (DeepManeuver [36], DeepBillboard [27], Perturbation Attack [43], FGSM [30], UniEqual) in terms of key evaluation metrics—Mean Error and Success Rate. Notably, UniAda achieves the highest mean error in steering (29.2° for CILRS, 25.6° for CILR, and 3.54° for MotionTransformer), which is 6.6°, 8.6° and 2.93° higher than the single-objective state-of-the-art method DeepManeuver. Furthermore, compared to multi-objective techniques (i.e., Perturbation Attack, FGSM, and UniEqual), UniAda emerges as the most effective solution for both objectives on average for all three models in almost all cases.

1.2 Test Case Transferability Analysis

Recent research on ADS testing has predominantly focused on generating adversarial and synthetic examples to trigger system misbehavior [27, 28]. Adversarial attacks in E2E ADSs can be generally categorized into white-box and black-box attacks [11, 4, 36]. White-box attacks involve full model access, while black-box attacks operate with limited query access without knowing the model parameters. Both types of attacks have been extensively explored for steering models (E2E ADSs predicting steering control) [4, 27, 11]. Black-box attacks are comparatively more practical in real-world scenarios due to their minimal information requirement [44], though generating effective adversarial samples remains challenging in such cases [45]. To address this issue, a line of research has proposed attacking a surrogate white-box model in which we know the parameters to generate adversarial samples, and leveraging cross-model transferability to attack black-box models for the same task [46–48]. Cross-model transferability (hereafter as transferability) involves using adversarial perturbations generated with a white-box model to attack black-box models [49]. Several studies demonstrate poor transferability of the white-box generated adversarial samples for steering models, but did not explore methods to enhance it [50, 46].

Many works have proposed transferability-enhancement techniques and validated their effectiveness on image classification tasks (e.g., ImageNet dataset) [51, 47, 52] and perception tasks (e.g., object detection) [53, 54]. Two commonly used approaches are input transformation enhancements [47, 51] and attack objective enhancements [55, 56]. Input transformation enhancements involve using an ensemble of transformed and original images (e.g., resizing) to generate the adversarial perturbation, thereby preventing overfitting to the white-box model. Attack objective enhancements apply different objectives to generate more transferable adversarial samples, such as disrupting model-agnostic features. However, there is limited research exploring methods to enhance attack transferability and assess the effectiveness of transferability-enhancement techniques on E2E ADSs.

Therefore, there are **two main challenges** yet to be solved in the current research: (1) How to improve the transferability of adversarial attacks on the E2E steering regression task? (2) Can the existing transferability-enhancement techniques designed for other tasks be implemented for the E2E steering regression task, and how effective are they? This part of the world seeks to fill the gap in autonomous driving research by conducting an empirical study on different transferability-enhancement techniques for

steering models and identifying the most effective ones. We summarize the following findings:

1) The transformation *Resizing+Translation* achieves the best black-box transferability, producing a 0.28° to 9.39° improvement in MAE compared to the basic framework (baseline).

2) The objective *Pred+Attn* achieves high effectiveness in both black-box and white-box attacks, with increases of up to 5.55° (white-box) and 6.21° (black-box) in MAE compared to the baseline.

3) The objective *Feat+Attn* is not recommended for either attack type, showing a decrease of up to 4.51° (black-box) and 66.09° (white-box) in MAE compared to the baseline.

4) The best transformation *Resizing+Translation* (0.06s) for transferability enhancements takes only half the runtime compared to the best objective *Pred+Attn* (0.11s) on average.

Furthermore, based on our results, we also provide insights to pave the way for future advancements in attack transferability within the field.

1.3 Test Case Validity Analysis

With the rapid development of deep learning (DL), many software applications have integrated deep neural networks (DNNs) to improve their performance. Despite the notable achievements of DL-based models across various domains [57–59], research has indicated their vulnerability to malicious attacks, necessitating comprehensive testing [27, 4, 8, 11].

Thorough testing of a DL system requires diverse and valid test inputs that can trigger model misbehavior. While plenty of Test Input Generators (TIGs) [8, 27, 4, 28, 35, 60, 7] have been proposed to efficiently generate error-inducing inputs, most of them fall short in checking the validity of the generated test inputs.

Input validation is a fundamental concept in software testing. In traditional software engineering, developers must create valid test cases to effectively evaluate the functional logic of the software and ensure compliance with specifications. When an invalid test case is encountered, the software will handle the input with errors and exhibit unexpected behavior. In such cases, the functional logic cannot be adequately tested, rendering those test cases ineffective [61–63]. Generally, enforcing input validation in software systems ensures only valid inputs are accepted for processing [62].

For DNNs, trained models are expected to generalize beyond the training data but within the valid input distribution [6]. However, DNNs often struggle to clearly distinguish between valid and invalid inputs, and they may still make predictions (possibly erroneous) for invalid inputs. For example, MNIST classifiers can predict digit labels for grayscale cat images with the same shape as MNIST samples, but these images are not meaningful in testing such classifiers. In DL testing, this brings difficulty in generating test inputs to effectively and correctly test the generalized behavior of a DNN. Unfortunately, research has demonstrated that existing TIGs may occasionally produce invalid inputs [11, 64, 65, 6].

To perform input validation for DNNs, expert labeling provides the most accurate but time-consuming assessment. To alleviate this burden, automated Input Validators (IVs) have been leveraged to automatically validate the input sample, sometimes referred to as anomaly detectors or out-of-distribution detectors [66, 67]. Different IVs identify invalid inputs (i.e., anomalies) through various methodologies. A straightforward method is to model the in-distribution density and classify out-of-distribution data as invalid, termed as density-based methods [68–71]. Another category, dubbed reconstruction-based methods, relies on the quality of the reconstructed image, such that hard-to-reconstruct samples are deemed invalid [72–76]. Distance-based methods calculate the feature-space distance [77–79] between the in-distribution (valid) and out-of-distribution (invalid) data.

A few testing techniques [11, 65] embedded an IV into their testing framework, aiming to automatically detect invalid inputs. Dola *et al.* [64] incorporated a variational auto-encoder (VAE)-based IV into the test generation framework to filter invalid inputs. Riccio *et al.* [6] examined the performance of two reconstruction-based IVs. However, they only studied the performance of a single type of input validator. This limitation leads to two questions: *(1) Does the results generalize to different types of IVs? (2) Do we select the right combination of TIG and IV for the DNN models under test? (i.e., can the selected IV accurately detect invalid inputs)?* **Our solution:** We include the assessment of four IVs across three types (i.e., reconstruction-based, density-based, and distance-based) to evaluate the compatibility across all combinations of TIGs and IVs, and incorporate the best IV into the testing framework (i.e., highest accuracy with human assessment).

Similar to traditional software testing, various DL-based test selection metrics have been proposed to quantitatively measure the novelty and diversity of the existing test suites [80]. For example, neuron coverage (NC) metrics [4, 5] measure the ratio of neurons that are activated in the DNN under test by the given test suite. Surprise

Adequacy (SA) [15] has also been proposed to measure the surprise of test input against the training set by monitoring the activation traces. Additionally, uncertainty quantifiers [81, 82, 13] measure the uncertainty of the DNN model when making decisions. Some works use these metrics as the testing objectives to generate test inputs [4, 7, 8]. However, to the best of our knowledge, their associations with the validity of test inputs are not systematically studied, making their uses in test input generation incur significant threats. For instance, Dola *et al.* [64] studied NC achieved by the generated test inputs and its association with validity by using the labels provided by automated IVs. This raises two potential issues: *(1) The generalization to other metrics is underexplored. (2) Validity labels assigned by automated IVs may not be reliable.* **Our solution:** We evaluate six test selection metrics in three types (i.e., NC, SA, and uncertainty quantifiers), and use human annotation as validity labels to systematically study their reliability.

To enhance the degree of validity in test input generation, we improve a typical testing framework by incorporating the distribution-aware objective of the best IV identified by our empirical study. The improvement in the number of valid inputs can reach up to 10% by human assessment.

1.4 Ensemble-based Test Selection

The susceptibility to errors of DL-based systems, particularly under adversarial inputs or out-of-distribution (OOD) scenarios, poses significant safety and reliability challenges [10, 4, 46]. The New York Times reported an incident in which a self-driving Uber vehicle fatally struck a pedestrian in Arizona because of an incorrect decision made by the autopilot system [83]. This underscores the necessity of conducting comprehensive testing to identify and mitigate potential faults in DNN systems.

When testing DL-based systems, test suites can become extremely large, encompassing potentially infinite and often unlabeled data. A comprehensive approach to testing a DL-based system would involve inputting all possible test suites to activate every feasible logic within the system and identifying any misbehavior or faulty logic [81]. However, manually labeling all data and executing the full test suite is impractical in terms of both labor and time consumption. To address these challenges, test selection metrics have been proposed to prioritize test inputs that are more likely to cause failures (e.g., mispredictions) and are underrepresented during training. This approach accelerates fault detection and performance improvement of

DNNs [13, 4, 5, 15, 18, 84]. Existing metrics encompass neuron coverage-based [4], uncertainty-based [13, 14], surprise adequacy-based [15], etc. Test input selection and prioritization using test selection metrics can be examined on both in-distribution (ID) and out-of-distribution (OOD) samples. Many previous works have followed this protocol and examined the performance of test selection metrics [81, 18]. Samples that follow the same distribution as the training data are regarded as ID data, in which DNN usually demonstrates high accuracy on them. OOD data are caused by distribution shifts, from either synthetic shifts [7, 4] (e.g., maliciously crafted perturbations) or natural shifts [85] (unseen and unperturbed new data, e.g., data collected in different periods of time, data collected from different sources). Prioritizing ID data can help to examine which part of the feature is insufficiently learned during the training process. Prioritizing OOD data can help to assess the generalizability of the trained model and discover faults not represented in the original training and test data.

The effectiveness of the proposed metrics is typically evaluated through two aspects: the capability to detect faults [81, 13] and the capability to guide retraining [4, 18, 86]. Although these metrics have shown promising results, we observe three major limitations in the existing literature, which we aim to address in this part of the thesis:

(1) Lack of consideration of consensus decision: We observe that the existing metrics select test inputs based on individual decisions (i.e., decisions of a single metric), and do not consider the consensus decision. Different metrics represent distinct aspects of a test input (e.g., DNN's neuron coverage, DNN's uncertainty to a test input, and DNN's surprise to an input). Selecting inputs that consider various aspects together can be beneficial as it helps mitigate biases or the incompleteness of individual scores caused by specific cases. For example, an input with low neuron coverage may result in high prediction uncertainty computed from the softmax layer. In this case, if the neuron coverage alone is used as the test selection metric, such input would not be prioritized during DNN repair/enhancement despite its high likelihood of exposing model misbehavior, which can ultimately lead to inefficient testing. However, this ensemble-based testing approach that considers the consensus decision is still underexplored in the current literature on DL testing.

(2) Insufficient effectiveness evaluation: It is important to consider comprehensive aspects to compare different test selection metrics. The capability to detect diverse faults in DNNs and the capability to guide retraining are two important aspects to assess the quality of the selected test suite, which is composed of the prioritized inputs by the metric [81, 18, 16]. Regarding the former factor, we denote a *fault* in

DNNs as the root reason that causes DNN mispredictions, e.g., multiple mispredictions may be attributed to the same fault. In the previous work, Average Percentage of Fault-Detection (APFD) [87] has been used to assess the fault detection capability based on the number of mispredicted inputs [13, 81]. This criterion evaluates the misprediction-detection rate and favors metrics that detect the highest number of mispredictions as early as possible. However, this approach can be misleading when multiple mispredictions are attributed to the same underlying cause/fault, as it fails to assess a metric's capability to prioritize mispredicted inputs associated with diverse causes [88]. As for the latter factor, the capability of guiding retraining is typically evaluated based on the improvement in accuracy after retraining using the selected test suite. However, the extent of this improvement varies depending on the test set used for evaluation, potentially overlooking certain faults. For example, a selected test suite might expose five DNN faults, but the evaluated test set may only reveal three of them. As a result, the discovery of the remaining two faults would not be captured by this criterion. Many previous studies have exclusively used retraining and/or APFD to compare different metrics [18, 86], which consequently leads to an insufficient and potentially biased evaluation.

(3) Limited task/dataset studied: The effectiveness of test selection metrics should be examined in a wide range of domains. We observe that most existing work conducted analysis limited to simple image datasets, such as MNIST [89], SVHN [90], CIFAR-10 [91]. They are mostly learned by CNN models and consist of similar feature representations. Although DeepXplore [4] also examined their test selection metric on driving and Android malware, their experimental settings (e.g., metric comparison, OOD construction) are relatively narrow. Dat [18] conducts an empirical study on diverse test selection metrics on both image and text data. However, their comparison of these metrics focuses solely on retraining performance, which may lead to a limited evaluation.

To address these shortcomings, we conduct a comprehensive empirical study on five test selection metrics and propose a novel testing framework for driving and malware detection datasets. For the empirical study part, we evaluate the metrics using two criteria: the capability of fault detection and retraining guidance. Instead of APFD, which measures the misprediction-detection rate, we adopt a more effective clustering-based approach that measures the rate of detecting distinct causes of the mispredictions [16]. The clustering-based approach assumes that each cluster (clustering from the misprediction set) corresponds to a fault, as mispredicted inputs within the same cluster are assumed to be mispredicted for the same reason. As

for the second criterion, to comprehensively evaluate the retraining performance, we leverage more than five OOD operators to construct the test set, each with 11 different distribution shifts, which is expected to simulate a generalized OOD scenario and are also more likely to trigger diverse DNN misbehaviors. Based on our empirical findings, we identify multiple best-performing metrics based on two criteria and propose an ensemble-based testing framework to extract a consensus decision for input selection. Validated on two Android malware datasets (AndroZoo [92] and Drebin [2]) and two real-world driving datasets (Dave [42] and Udacity [26]), our observations reveal that none of the five studied metrics achieve statistically significant top performance across both evaluation criteria simultaneously. In contrast, our method successfully achieves this. Compared to the best baseline in retraining guidance, our method achieves at least the same amount of accuracy improvements while delivering substantial and statistically significant improvements in fault detection (e.g., detecting 5 extra faults on average, with statistical significance in 75% of the cases). The same holds true for the best baseline of fault detection. Additionally, our ablation study statistically validates the effectiveness of each component of our method, including scale unification and ensemble-based selection.

1.5 Contributions of Thesis

This thesis makes significant contributions to improving the reliability and safety of Deep Learning (DL) based systems through a suite of novel, automated testing methodologies. We propose three core testing approaches: UniAda, validity-aware testing, and ensemble-based testing, complemented by a comprehensive empirical study on test case transferability. Collectively, these contributions to test case generation and selection provide practical tools and insights for uncovering critical faults in the SUT, ensuring the realism of tests, and enabling more efficient debugging and improvement of DL models, particularly in safety-critical domains like autonomous driving.

- In the domain of test case generation for ADSs, we propose UniAda, a novel attack technique for DL-based E2E ADSs that can generate multi-objective universal adversarial perturbations for the input driving video. UniAda is effective in triggering errors on three state-of-the-art DL-based E2E ADSs for both steering and acceleration controls, tested with both simulated and real-world

data. UniAda improves reliability by proactively exposing a common feature pattern of test cases that can induce critical, previously overlooked faults.

- We also propose a new strategy, Adaptive Weighting Scheme (AWS), employed in UniAda for balancing different objectives in a real-time manner, which maintains consistent training rates for each objective. The effectiveness of AWS is validated by comparing UniAda with the equal-weighted counterpart, UniEqual.

- The experiment results show that UniAda outperforms five benchmarks across two metrics, inducing steering and speed deviations from 3.54° to 29° and 11 km/h to 22 km/h on average. All the implementations are available at https://github.com/UniAdaRepo/UniAda/.

- For test case transferability analysis, we systematically assess the effectiveness of various transferability-enhancement techniques for E2E steering models. The generated test cases are likely to fail not just one model, but an entire family of models. This breaks the dependency on white-box access and allows for more rigorous, independent safety auditing, a cornerstone of reliable system integration. We provide insights into the superior performance of these techniques and open-source our implementation at https://github.com/EnhanceRepo/EnhanceTransferability.

- We identify nine enhancements in total–three input transformations and six attack objectives–applicable to E2E steering models. We evaluate their effectiveness in both white-box and black-box attacks using two datasets and four steering models. For black-box and timely attacks, *Resizing+Translation* is recommended. For high effectiveness in both attack types, *Pred+Attn* is recommended. However, we do not recommend *Feat+Attn* for either attack type.

- In test case validity analysis, we analyze the association of six test selection metrics with the test input validity. Our results indicate that, in comparison to neuron coverage and surprise adequacy, uncertainty quantifiers offer a more accurate and positive association with input invalidity. Validity-aware testing improves reliability through efficient testing and, hence, efficient system fault correction. Meanwhile, this ensures resources are spent on meaningful test cases.

- Building upon our empirical findings, we enhance the existing testing framework by incorporating distribution awareness through eight different joint optimizations. The results have demonstrated a 2% to 10% increase in the number of valid inputs by human assessment. The implementation and data are available at https://github.com/InputValidityRepo/InputValidity.

- As for the ensemble-based testing, we propose a novel scale and granularity unification strategy within our testing framework to unify different test selection metrics, facilitating ensemble-based testing. Our method is statistically effective in both fault detection and performance improvement of the SUTs across autonomous driving and malware detection domains. Ensemble-based testing improves the system's reliability by making the retraining process of system enhancement more efficient and effective using critical test inputs. The implementation and data are available at https://github.com/EnsembleTesting/EnsembleTesting/.

- For evaluation, we utilize two key criteria: the capability of retraining guidance and the capability of fault detection. Specifically, we adopt a novel clustering-based approach to more effectively estimate the cause of DNN misbehaviors, moving beyond the conventional focus on the number of mispredictions.

To facilitate understanding of the contributions of the thesis, we also provide a high-level summary table that compares existing work and our studies. As illustrated in Table 1.1, we have made meaningful contributions on each studied topic in the thesis.

As for the first topic, effective test case generation in autonomous driving, the existing work mainly generates test cases that only induce misbehavior on the vehicle's steering control, neglecting other important maneuvers. Also, their generated test case (e.g., adversarial perturbation) is mostly effective on single image input, i.e., image-specific test cases. Our proposed method addresses these two main limitations. The proposed UniAda is capable of generating test cases that induce misbehavior on both steering and speed controls. Moreover, our generated test case is image-agnostic, such that a single perturbation can be applied to multiple images and render them effective in inducing model misbehavior on both maneuvers.

As for the second topic, transferable test case generation in ADSs, the existing work generates a test case that is effective for one targeted model only. It becomes ineffective when we want to use it on other models. Our empirical study address this

Table 1.1 Contributions of the thesis

Topic	Existing work	Our method / study
Effective test case generation in ADSs	Steering misbehavior; Image-specific test case	Steering & speed misbehavior; Image-agnostic test case
Transferable test case generation in ADSs	Model-specific test case; (work only on one model)	Model-agnostic test case; (i.e., transferable)
Valid test case generation (multiple domains)	Invalid test cases; (not real and meaningful)	Valid test case generation; (incorporate validity checker)
Test case selection (multiple domains)	Inefficient test cases selection (single aspects) Slow DNN improve	Ensemble-based test selection (diverse aspects) Faster DNN improvement

problem by studying transferability-enhancement techniques that help in generating a transferable test case that works on multiple models. In the real world, the tester rarely knows the internal details of the targeted AI system. Therefore, transferable test cases that work across different models are helpful.

As for the third topic, valid test case generation, the existing work often generates invalid test cases. It is not meaningful to use non-real or invalid test cases to test a system. Our proposed method leverages a validity checker to realize valid test case generation. Meanwhile, we also make recommendations on automated validity checkers for researchers and practitioners in the field of software engineering. For example, which is more effective on complex image data?

As for the last topic, test case selection, the existing work pick test cases based on a single aspect (e.g., a single score). This is inefficient to select test cases for fast model improvement. Our proposed method utilizes ensemble-based test case selection, which combines multiple methods to pick better test inputs using a consensus approach. This novel method finds tests that reveal more diverse and important model faults, assisting in more efficient model performance improvement. This helps testers save time and cost in DL testing.

1.6 Organization of the Thesis

The remainder of this thesis is structured as follows:

Chapter 2 provides a comprehensive review of related work in the thesis, covering topics such as adversarial attacks for DL-based ADSs, test case transferability, test input generation and validation, and ensemble learning for test case selection.

Chapter 3 introduces UniAda, a multi-objective white-box attack technique with a core function that revolves around crafting an image-agnostic adversarial perturbation capable of simultaneously influencing both steering and speed controls. UniAda capitalizes on an intricately designed multi-objective optimization function with AWS, enabling the concurrent optimization of diverse objectives.

Chapter 4 presents transferable attack analysis for ADSs, a comprehensive empirical study of nine transferability-enhancement methods on E2E ADSs, covering two types: three input transformation enhancements and six attack objective enhancements. We evaluate their effectiveness on two datasets with four steering models.

Chapter 5 describes test case validity analysis, which empirically studies four automated Input Validators (IVs) and six test selection metrics. We also enhance existing testing frameworks by incorporating distribution awareness through joint optimization based on the empirical findings.

Chapter 6 describes a novel ensemble-based testing framework that integrates scale and granularity unification to systematically combine multiple test selection metrics. Extensive experiments demonstrate the superior performance of our framework compared to baselines.

Finally, Chapter 7 concludes the thesis by summarizing the research findings, highlighting the main contributions, and suggesting avenues for future work.

Chapter 2

Literature Review

In this chapter, we delve into a literature review describing different DL testing tasks studied in this thesis, such as adversarial attacks for DL-based ADSs (test case generation), transferability of the generated adversarial samples, test case validation, test case selection, and ensemble learning.

2.1 Adversarial Attacks in DL-based ADSs

2.1.1 DL in Autonomous Driving

The integration of DL into E2E ADSs [93–95] traces back to the late 1980s, Pomerleau *et al.* [96] built the Autonomous Land Vehicle in a Neural Network (ALVINN) system, which uses a shallow three-layer network. Later in 2016, Bojarski *et al.* [1] trained a Convolutional Neural Network (CNN) to directly map raw image pixels from a single front-facing camera to steering commands. Similarly, in the context of the Udacity self-driving car challenges [26], researchers embarked on constructing cutting-edge agents that interpret sensor inputs to govern vehicle operations. Teams achieving state-of-the-art performance were ranked on the Udacity leaderboard [1]. Many successful research works have demonstrated the power of the E2E learning approach in DNN-based ADSs, and this evolution continues unabated. More recently, Codevilla *et al.* [25] introduced an innovative conditional imitation learning framework, which takes RGB images, speed measurements and navigation instructions as inputs, generating steering and acceleration-related car controls as outputs.

[1] https://github.com/udacity/self-driving-car/tree/master/challenges/challenge-2

2.1.2 Adversarial Attacks in E2E ADSs

Prevalent adversarial attacks in autonomous driving can be broadly categorized in white-box and black-box attacks. Within each attack type, the attacker can conduct either online or offline attacks.

Among these attack types, offline white-box attacks poses the most significant threat, due to the full access to the SUT and potential iterative interactions with it [97]. Specifically, offline attacks [98, 30, 99, 43] apply the adversarial perturbation on a pre-acquired fixed dataset. For example, for an ADS, the prediction made on the current adversarial sample will not affect the perceived image input on the next timestamp. In white-box attacks [62, 100], information regarding internal model structures and parameters are available for the attacker, which enables the attacker to formulate highly targeted and effective adversarial perturbations to manipulate the model's behavior.

Numerous offline white-box attack techniques has been proposed for ADSs. Many of these approaches are designed to generate adversarial or transformed images as sensor inputs, to trigger erroneous maneuvers [27, 11, 4, 28, 34]. For instance, DeepXplore [4] employs neuron coverage and differential behavior-based metrics within a white-box testing framework to create transformed images (e.g., different brightness) that mislead the steering angle predictions of ADSs.

Apart from image-specific transformations, in which they apply distinct transformations to different images (i.e., a single transformation is ineffective across multiple images), some studies [27, 28, 101, 36] have pushed beyond this constraint by seeking transformations capable of consistently inducing model misbehavior, affecting sequences of images (i.e., driving videos). Notably, Von *et al.* (DeepManeuver) [36], Zhou *et al.* (DeepBillboard) [27] and Kong *et al.* (PhysGAN) [28] utilizes variants of prediction differences as target metrics to generate a universal adversarial perturbation. These methods identify a universal perturbation pattern that can persistently provoke misbehavior on steering predictions across a sequence of driving images. However, their focus is confined to specific testing scenarios involving billboard appearances and targeting steering angle misbehavior exclusively. Beyond steering angle, speed-related controls bear equal significance in driving scenarios. Moreover, DeepManeuver exclusively performs attacks within a simulator, lacking analysis in real-world scenarios. The existing limitations of the previous approaches result in the incompleteness of comprehensive testing of modern ADSs.

2.2 Attack Transferability

Although white-box attacks usually lead to an effective attack performance, it is sometimes difficult for the attacker to have access to the internal information of the SUT. Under such scenarios, utilizing the transferability of the generated adversarial samples to conduct black-box attacks is a preferable option.

Black-box attacks involve scenarios where the attacker is limited only to the query access to the model, knowing only the model's output for a given input [102]. For example, DeepRoad [11] implemented a black-box variational-autoencoder-based method that applied various weather transformations to craft adversarial images that mimic real-world weather conditions. The adversarial samples can induce misbehavior in the steering model. Some works also utilized the transferability of white-box-generated adversarial examples to perform black-box attacks. For instance, Deng *et al.* [50] discovered that adversarial examples generated with optimization-based and generative-network-based methods for E2E steering models perform poorly on other black-box models. Zhang *et al.* [46] observed similar results for multi-output E2E ADSs and pointed out several research directions to enhance transferability, however, they did not validate them experimentally. Overall, many existing research on E2E ADSs has focused on improving the effectiveness of white-box attacks, while enhancements for black-box attacks remain largely underexplored. Notably, there is a lack of research on techniques that enhance the attack transferability on black-box models for E2E ADSs.

The cross-model transferability of adversarial examples generated with the white-box model can be used to attack black-box models for the same task [51, 54, 48]. Specifically, the attacker crafts the attack against a white-box surrogate model, which is expected to have a similar architecture as the target SUT (i.e., black-box model), hoping that the attack will be effective on the target SUT. However, directly transferring the generated adversarial perturbation to black-box models usually results in poor performance due to overfitting to the white-box model [50]. The transferability property of an attack is satisfied when an attack developed for the surrogate model is also effective against the target model. Early studies observed the transferability property on adversarial samples [10, 103]. Then, previous work has reported empirical findings about the transferability of evasion attacks [104–106, 39, 44, 107, 108].

Meanwhile, various methods have been proposed to improve the transferability of the attack in the field of image classification. Common methods include but are not limited to applying various input transformations [47, 51], attacking model-agnostic

features [109, 110], and attacking model-agnostic semantic properties (e.g., attention) [52, 111]. For example, Xie *et al.* [47] proposed to improve the transferability of adversarial examples by creating diverse input patterns, while Dong *et al.* [51] used a set of translated images to optimize an adversarial sample. Similarly, in the field of perception, several studies have expanded on these concepts to enhance the transferability for object detection or semantic segmentation tasks [112, 53, 54]. For example, Lu *et al.* [53] and Hashemi *et al.* proposed to enhance transferability by attacking intermediate feature maps for semantic segmentation and object detection tasks.

The proven effectiveness and importance of transferable adversarial samples in other fields encourage us to address the research gap regarding transferability analysis in the steering task for E2E ADSs.

2.3 Test Case Validation

The typical workflow of test input validation, as shown in Figure 2.1, consists of two phases: 1) test input generation; 2) test input validation. By feeding seed inputs selected from the dataset, TIGs, according to their optimization objectives and DNNs under test, generate a misbehavior-inducing test input for each corresponding seed. To effectively and efficiently reveal the generalization errors of DNNs under test, each generated test input should be assessed for validity. The validity assessment can be performed by either automated IVs or domain experts.

2.3.1 Existing TIGs.

Diverse TIGs [4, 11, 8, 27, 7] have been proposed to produce misbehavior-inducing inputs to the DL system under test. They can be broadly categorized into three main approaches based on how they handle input samples [6]: (1) Raw Input Manipulation (RIM), (2) Generative DL Models (GDLM), and (3) Model-based Input Manipulation (MIM). Studies in (1) [4, 7, 27] applied pixel-level perturbations to the original inputs based on the gradients computed with respect to a testing objective, such as adversarial attacks and input transformations (e.g., adding black dots, weather transformations). For example, DeepXplore [4] generates artificial inputs by solving a joint optimization problem with two objectives: maximizing the behavioral differences between multiple models and maximizing the number of activated neurons. Pei *et al.* [4] has shown that the test inputs generated by DeepXplore are effective at revealing model misbehaviors

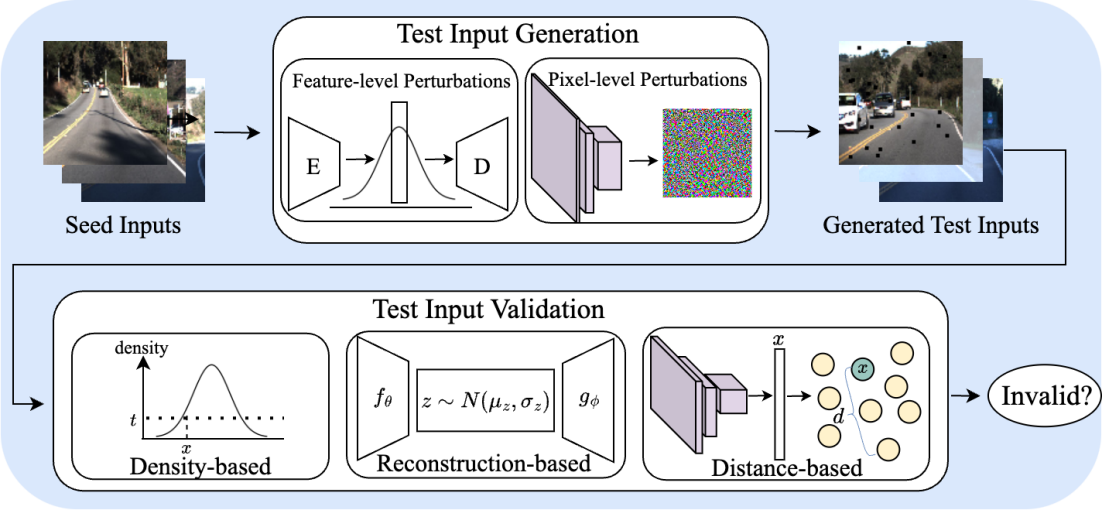


Fig. 2.1 Workflow of Input Validation: 1) Test Input Generation from seed inputs; 2) Test Input Validation by automated IVs.

(e.g., misclassification). Moreover, retraining the SUT with test inputs generated by DeepXplore increases the prediction accuracy of the SUT, and the improvement is superior compared to retraining with inputs generated by simple adversarial techniques (e.g., FGSM [10]). Similarly, DLFuzz [7] leveraged differential fuzzing testing to expose incorrect behaviors of DL systems. They minutely mutated the input to maximize the NC and the prediction difference between the original input and the mutated input. Moreover, DeepCT [113] generates artificial tests to cover as many t-wise interactions as possible. DeepHunter [114] is a fuzzing-based test generation algorithm to hunt defects in DL models. The fuzzing is guided by the coverage metrics defined in DeepXplore [4] and DeepGauge [5].

For (2), some [8, 115, 116] also conducted perturbations in the feature space, aiming to generate more diverse inputs. For instance, Sinvad [8] trained a Variational Autoencoder (VAE) and modified the input feature space. The test cases are obtained by decoding the perturbed feature. Dunn *et al.* [115] leveraged generative models to generate inputs that vary in their high-level features, such as different object shapes, locations, and colors.

For (3), the original input is transformed into an input model instance that abstracts its main features. The manipulation is performed on the model, which is finally transformed back to the original format. DeepJanus [117] is an example of such an approach, which abstracts a model instance from the bitmap, obtaining a Scalable Vector Graphics (SVG) representation of handwritten digits (MNIST dataset, used

in their experimentation). DeepJanus can directly modify the model instance and transform it back to the original input format by means of a rasterisation operation.

2.3.2 Existing IVs.

For DNNs, trained models are expected to generalize beyond the training data but within the valid input distribution [6]. However, DNNs often struggle to clearly distinguish between valid and invalid inputs, and they may still make predictions (possibly erroneous) for invalid inputs. For example, MNIST classifiers can predict digit labels for grayscale cat images with the same shape as MNIST samples, but these images are not meaningful in testing such classifiers. In DL testing, this brings difficulty in effectively and correctly testing the generalized behavior of a DNN, as the generated test inputs by TIGs might be invalid, as demonstrated in [11, 64, 65, 6].

An input is deemed *invalid* if it is anomalous or an outlier to the data distribution learned from training. Although an invalid input satisfies the acceptable input tensor shape of a DNN model, it might be semantically irrelevant to the given task. Due to the scarcity of representative invalid/anomalous data, a common practice [11, 6, 64] is to apply semi-supervised learning, which utilizes the information from valid/normal data only [66]. Semi-supervised techniques can be generally categorized into three types [67, 118]: statistical density-based, reconstruction-based, and distance-based methods.

Statistical Density-based Methods explicitly estimate a statistical probabilistic model for valid data and identify low-density data as invalid [71, 119, 69, 70]. *Reconstruction-based Methods* rely on the reconstruction of images, which first maps the original input image to the latent space and then reconstructs the original input by finding an inverse mapping for the latent vector [72–76]. A valid sample is expected to produce a good reconstruction with low reconstruction errors. *Distance-based Methods* classify a sample as invalid if the sample is located far away from the valid data in the feature space [120, 121, 77, 122–126].

To assess the validity of test inputs generated by testing techniques, Zhang *et al.* [11] utilized a distance-based IV to validate the test input images for autonomous driving systems. Stocco *et al.* [65] proposed a self-assessment oracle to monitor the confidence of input images by implementing reconstruction-based validation techniques. Similarly, Dola *et al.* [64] validated the test inputs by two existing IVs. However, they do not assess the accuracy of selected IVs. Recently, Riccio *et al.* [6] conducted an empirical study to examine the effectiveness of two reconstruction-based IVs in validating the test inputs generated by different TIGs, however, they do not

analyze the generalization to other types of IVs. Also, the effectiveness of test selection metrics (described in the next section) in filtering invalid inputs is underexplored.

2.4 Test Case Selection

Owing to a large amount of automatically generated test inputs, test selection metrics have been proposed to give priority to test inputs that are more likely to detect misbehavior exhibited by the DL system under test [18, 127]. Main categories of test selection metrics include NC-based, SA-based, and uncertainty-based metrics.

For each test input, NC [4] measures the boolean coverage of every neuron according to its activation in the DNN under test. It can be leveraged to guide the testing to explore different and diverse incorrect DNN behaviors. However, Harel *et al.* [128] argued that high NC impeded the generation of effective test suites in terms of naturalness and misbehavior detection effectiveness. Recent work [13, 5] introduced multiple variants of NC based on different coverage profile, such as k-Multisection Neuron Coverage, Neuron Boundary Coverage, and Strong Neuron Activation Coverage.

SA-based metrics, proposed by Kim *et al.* [15], measure the relative novelty or surprise of a new test input compared to the training data by recording the activation trace of the input. More precisely, the degree of surprise measures the dissimilarity of the neurons' activation values when confronted with this new input with respect to the neurons' activation values when confronted with the training data. In other words, a good test suite should range from very similar to very different inputs to those observed during training. They hypothesize that a set of test inputs is preferable for both testing and retraining when it covers a diverse range of surprise values. They also demonstrated that SA was effective in detecting adversarial examples.

In terms of uncertainty-based metrics, previous works [82, 13] introduced a variety of quantifiers, such as DeepGini [13], Vanilla-Softmax [129], Prediction-Conference Score [130], and Monte-Carlo Dropout [131]. For example, DeepGini [13] was proposed to prioritize the test inputs and select the most uncertain inputs. The authors have demonstrated that DeepGini is useful in both misprediction detection and retraining guidance. Gal *et al.* [131] proved that the variance of the softmax function resulting from neuron dropout is a good estimate for model uncertainty, and this metric was therefore leveraged to select test inputs. Kendall *et al.* [132] proposed a model to capture jointly aleatoric uncertainty (originating from noise inherent to the observations) and epistemic uncertainty (induced by the fact that the model is

not trained on all possible data). The latter type is what is commonly referred to as model uncertainty, which can be captured by dropout variance. Test inputs with higher uncertainty are then selected for model enhancement.

There are also several hybrid selection metrics, which fuse different strategies based on the property of the test case. For example, Hu *et al.* [18] proposed a distribution-aware metric that utilized an OOD detector to distinguish and select separately from ID and OOD data. It selects the most uncertain samples from the ID data and performs random selection for the OOD data.

To evaluate test selection metrics, researchers typically assess their capability of fault detection and the capability of retraining guidance. For example, Weiss *et al.* [81] leveraged APFD as an indicator of DNN fault detection to assess different metrics. Specifically, when plotting the percentage of detection mispredicted tests against the number of prioritized tests, APFD is calculated as the area below the plotted line. Formally, following the existing work, for a permutation of N inputs in the selected test suite in which there are K inputs that will be mispredicted, let o_i be the order of the first test that reveals the kth mispredicted test. The APFD value for this permutation can be calculated as $APFD = 1 - \frac{\sum_{k=1}^{K} o_k}{KN} + \frac{1}{2N}$.

However, it has been pointed out to be ineffective since many mispredictions could be attributed to the same cause [16]. In this case, APFD cannot effectively assess how the selected test inputs can detect diverse faults. Later, Aghababaeyan *et al.* [16] proposed a more effective approach, which leveraged a clustering-based method on feature embeddings. Specifically, they group similar mispredicted inputs, aiming for each cluster to represent a common set of features that are the plausible cause for mispredictions, and treated each cluster as a unique fault. The authors also proposed using geometric diversity to guide DNN testing, demonstrating that it serves as a better indicator of DNN fault detection compared to coverage-based criteria. As for the latter factor, the capability of guiding retraining is typically evaluated based on the improvement in accuracy after retraining using the selected test suite.

2.5 Ensemble Learning

In the field of ensemble deep learning, there are several different ensemble strategies, which can be broadly categorized into decision fusion methods (e.g., majority voting, unweighted model averaging) and classical learning-based methods (e.g., bagging, boosting, stacking) [133, 134]. The former focused on how to aggregate the decisions of different DNNs, and the latter explored the training of multiple DNNs. Our work

builds on the former and proposes a novel fusion strategy on the outputs/scores of different test selection metrics to achieve a consensus decision. Previous research has demonstrated the effectiveness of ensemble learning in many fields, such as image classification and medical imaging [135–137]. For example, Pawlowski *et al.* [136] leveraged ensembles of DNNs by variational approximation for outlier detection on image data. Lakshminarayanan *et al.* [138] utilized deep ensembles for uncertainty estimation in DNN predictions, which has been shown to be better than approximate Bayesian DNNs.

Ganaie *et al.* [133] pointed out that majority voting is less biased compared to a particular base learner. Gopal *et al.* [137] showed an accuracy improvement by using a majority voting algorithm for the brain tumor classification problem.

However, to the best of our knowledge, few studies have investigated ensembles of test selection metrics. Also, most existing works lack a comprehensive and unbiased evaluation on the different capabilities of studied metrics.

2.6 Summary

In this chapter, we have presented the techniques most relevant to this thesis. For each subsection, we provide a summary of recent methodologies and highlight their limitations.

Chapter 3

UniAda: Universal Adaptive Multiobjective Adversarial Attack for End-to-End Autonomous Driving Systems

Adversarial attacks play a pivotal role in testing and improving the reliability of deep learning (DL) systems. Existing literature has demonstrated that subtle perturbations to the input can elicit erroneous outcomes, thereby substantially compromising the security of DL systems. This has emerged as a critical concern in the development of DL-based safety-critical systems like Autonomous Driving Systems (ADSs). The focus of existing adversarial attack methods on End-to-End (E2E) ADSs has predominantly centered on misbehaviors of steering angle, which overlooks speed-related controls or imperceptible perturbations. To address these challenges, we introduce UniAda—a multi-objective white-box attack technique with a core function that revolves around crafting an image-agnostic adversarial perturbation capable of simultaneously influencing both steering and speed controls. UniAda capitalizes on an intricately designed multi-objective optimization function with the Adaptive Weighting Scheme (AWS), enabling the concurrent optimization of diverse objectives. Validated with both simulated and real-world driving data, UniAda outperforms five benchmarks across two metrics, inducing steering and speed deviations from 3.54° to 29° and 11 km/h to 22 km/h on average. This systematic approach establishes UniAda as a proven technique for adversarial attacks on modern DL-based E2E ADSs. This work was published in the IEEE Transactions on Reliability in 2024 [46].

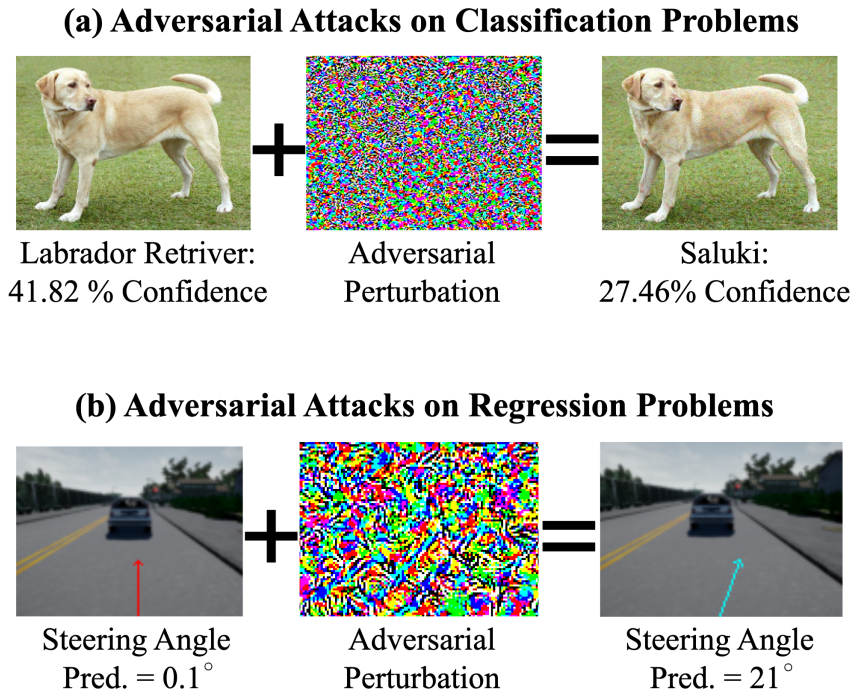


Fig. 3.1 Examples of adversarial attacks on image classification and regression tasks. Pixel values of perturbation τ are scaled for visibility.

The subsequent sections of this chapter are organized as follows: Section 3.1 provides theoretical foundations, while Section 3.2 introduces the threat model of this chapter. Section 3.3 introduces UniAda and its methodology. Sections 3.4 and 3.5 delve into the experimental setup and results, respectively. Section 3.6 discusses the transferability of UniAda and two baselines, limitations of UniAda, and threats to the validity of this work. A summary of this chapter is provided in Section 3.7.

3.1 Preliminaries

Adversarial Attacks: In this section, we provide a concise and formulated overview of adversarial attacks, covering both image classification and regression problems. Additionally, we elucidate the concept of universal perturbation.

In adversarial attacks, the generated perturbation τ can trigger errors in the model prediction. By adding a subtle τ to the input sample x (e.g., an RGB image), a model m trained with optimal parameter μ can make a different prediction (mostly wrong) from the original. Specifically, for image classification problems, we wish to find a vector

τ that can induce the CNN model to predict different class labels for x and $x + \tau$, satisfying:

$$m(x + \tau; \mu) \neq m(x; \mu) \quad s.t. \ \|\tau\|_p \leq \epsilon \tag{3.1}$$

where $\| \cdot \|_p$ indicates the L_p norm, and L_0, L_2 and L_{inf} are commonly used metrics [39, 30]. The constraint is to ensure $x + \tau$ and x are visually the same to human eyes. For regression problems, the goal of the adversarial attack is to find τ that maximizes the discrepancy between the two numerical predictions, satisfying:

$$\underset{\tau}{\text{argmax}} \ |m(x + \tau; \mu) - m(x; \mu)| \quad s.t. \ \|\tau\|_p \leq \epsilon \tag{3.2}$$

Normally, in regression problems, the attack is considered successful if the prediction difference is larger than a predefined threshold [50].

Fig. 3.1 gives examples of adversarial attacks on the image classification model (Fig. 3.1(a) with $\epsilon = 15$) and autonomous driving regression model (Fig. 3.1(b) with $\epsilon = 2$). Specifically, for the classification task, the CNN model is very vulnerable to a subtle perturbation, which changes its prediction from *Labrador Retriver* with 41.82% confidence to *Saluki* with 27.46% confidence. For the regression task, we show an example of a CNN-based E2E autonomous driving model that predicts the steering angle value from the image. The added imperceptible perturbation changes its prediction from 0.1° to 21°. The above example shows that CNN models are vulnerable to such attacks.

For a **Universal Perturbation** [39] (i.e., image-agnostic), the goal is to seek a numerical vector τ that can successfully fool almost all datapoints sampled from $\mathbf{X}$, where $\mathbf{X}$ denotes a distribution of data. In regression problems, a universal perturbation τ satisfies:

$$\begin{aligned} &|m(x + \tau; \mu) - m(x; \mu)| > \delta \\ &s.t. \ \|\tau\|_p \leq \epsilon \quad \textit{for most } x \in \mathbf{X} \end{aligned} \tag{3.3}$$

In our paper, $\mathbf{X}$ is a distribution of images, represented by a driving record. We attack three CNN-based autonomous driving models (introduced in Section 3.4.2) that perform a multi-output regression task, which takes an RGB image as input and outputs continuous numerical steering angle and acceleration prediction.

3.2 Threat Model

Our work focuses on attacking DNN-based autonomous driving models in the image regression domain. We consider four types of threat model in this work: adversarial falsification, adversary's knowledge, adversarial specificity, and attack frequency [97].

Adversarial Falsification: For an intentional attack, the attacker can design an adversarial example $(x + \tau)$ to fool the model m into making a wrong decision. However, this manipulated example is not visually different from x to a human observer. For an unintentional attack, the attacker can unintentionally send an input $\hat{x}$ that is sampled from a distribution different from the training distribution. Specifically, $\hat{x}$ has a label y, and $m(\hat{x}) \neq y$. In this case, the model fails on $\hat{x}$ due to the distribution shift.

Our goal is to perform *intentional* attack on a regression problem, which adds a subtle perturbation τ on the sample image x to cause misbehavior on the autonomous driving model m. The adversarial image is similar to the original image and conforms to the training distribution.

Adversary's Knowledge: Based on the available knowledge to the model, attacks can be categorized into white-box, grey-box and black-box. They correspond to full access, partial access, and no access to the model's internal structure and parameter information. In our paper, we perform *white-box* gradient-based attacks, where we know the model architecture, hyperparameters, and model weights. We generate adversarial examples by calculating model gradients with our designed objective function.

Adversarial Specificity: This includes targeted attacks and non-targeted attacks. For targeted attacks, the attacker misguides DNN to a specified prediction output (e.g., a specified predicted class for a classification problem, bigger/smaller predicted values for regression problems). In non-targeted attacks, the adversarial output can be arbitrary except for the original one.

We construct a *targeted* adversarial sample that follows the specified attack direction. For example, we ask UniAda to construct an adversarial sample that can induce the model to turn right (i.e., to predict a bigger steering value) instead of turning left.

Attack Frequency: This includes one-step and iterative attacks, which are distinguished by the number of queries they make to the victim model. We employ *iterative* attacks that progressively refine adversarial examples over multiple steps to achieve higher effectiveness.

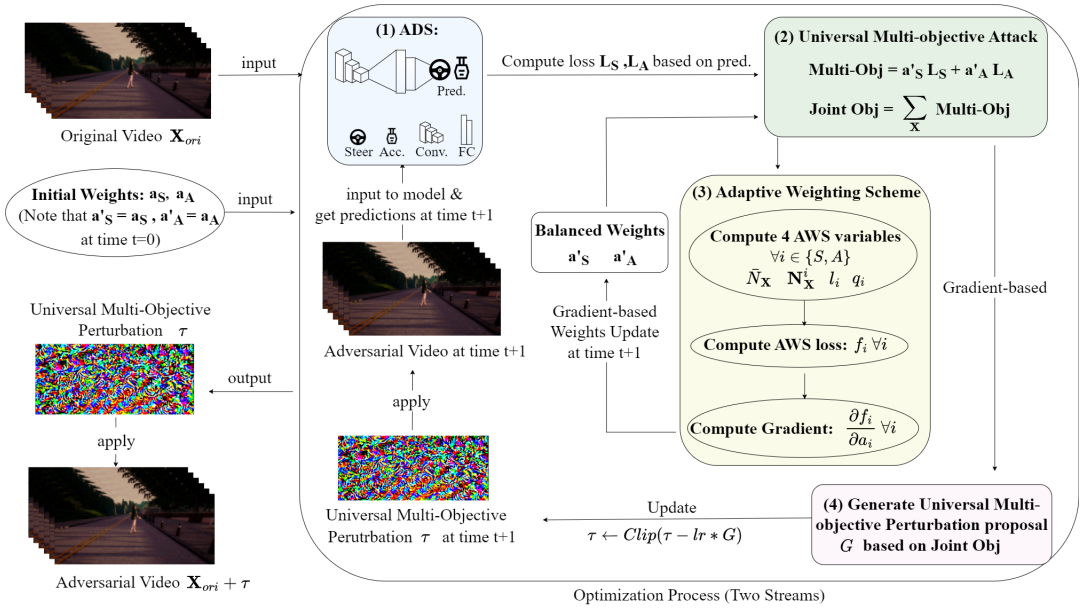


Fig. 3.2 UniAda Overview: Our attack technique contains two main optimization streams. The first one is the Universal Multi-objective Attack (in green) to generate a perturbation that is influential to multiple images and multiple car controls (i.e., steering S and acceleration A). The second one is the Adaptive Weighting Scheme (in yellow), which iteratively balances the objective weights to ensure similar training rates. Best viewed in color.

3.3 Methodology

3.3.1 UniAda Overview

This section outlines UniAda, which employs adaptive objective weights to create a universal adversarial perturbation. This perturbation consistently misleads the targeted model across multiple images, impacting several car controls simultaneously. The overview of UniAda is shown in Fig. 3.2. Given a driving video encompassing dozens of images and initial objective weights, UniAda commences the perturbation discovery through parallel optimization streams: Perturbation Optimization and Objective Weight Optimization. These streams evolve in tandem during each iteration step t.

In the perturbation optimization stream, each image inside the input video is fed into the autonomous driving model. This yields predictions for each car control, specifically steering angle and acceleration, as depicted in step *(1) ADS*. Utilizing these predictions, UniAda computes individual losses for each car control (L_S, L_A). Drawing from these losses and the prevailing objective weights a'_S, a'_A, UniAda in step *(2)* computes the joint optimization function (detailed explanations can be found in Section 3.3.2). Subsequently, employing a gradient-based methodology, UniAda derives the joint function's derivative with respect to the image, and the resultant gradient G serves as the current perturbation proposal. G is then leveraged to update the universal multi-objective perturbation for the subsequent time step $t + 1$.

Simultaneously, the weight optimization procedure starts, enabling the dynamic adjustment of weights for each objective. At a given step t, UniAda employs the ongoing objective weights a'_S, a'_A, coupled with the losses (L_S, L_A), to calculate four variables within step *(3) Adaptive Weighting Scheme*. These variables contribute to the AWS loss. Subsequently, UniAda computes gradients of the AWS loss f_i with respect to each objective weight a_i. These gradients facilitate a gradient-based update mechanism, resulting in refreshed balanced weights at the following time step. Details for Adaptive Weighting Scheme are elaborated in Section 3.3.3.

To generate universal multi-objective perturbations, both optimization processes iteratively proceed until the maximum number of epochs is reached.

3.3.2 Universal Multi-objective Attack

We design the loss function [28] for each objective i to maximize the difference between original and adversarial model predictions:

$$L_i(\mathbf{X}^n_{ori}, \tau) = \frac{1}{\beta} \exp\Big\{-\frac{1}{\beta}(m_i(\mathbf{X}^n_{ori} + \tau) - m_i(\mathbf{X}^n_{ori})) \times d_i\Big\} \quad \forall n \in \{1, ..., N\} \tag{3.4}$$

The hyperparameter $d_i = \pm 1$ is the attack direction for objective (car control) i, which is set to be $-1/1$ if we would like to attack in the negative/positive direction (adversarial prediction is learned to be smaller/larger than the original).

Notations encompass τ for perturbation at time t, $\mathbf{X}^n_{ori}$ signifying the n^{th} image frame inside the input video $\mathbf{X}_{ori}$, and $m_i(\mathbf{X}^n_{ori} + \tau)$ representing the prediction of autonomous driving model m for objective i from the perturbed adversarial image $\mathbf{X}^n_{ori} + \tau$. β signifies the sharpness parameter. N denotes the total number of images of the input driving record. Similar to DeepBillboard, we use the original prediction as the attack reference.

Notably, prior work solely concentrated on perturbations inducing deviations in the model's steering angle predictions, disregarding speed-related car control objectives. To generate a universal perturbation capable of concurrently influencing multiple car controls, we devise a multi-objective loss as follows:

$$O(\mathbf{X}^n_{ori}, \tau) = \sum_{i=1}^{C} a_i L_i(\mathbf{X}^n_{ori}, \tau) \quad \forall n \in \{1, ..., N\} \tag{3.5}$$

where C denotes the number of objectives, and parameter a_i reflects the weight assigned to objective i. In our case, there are two objectives, steering and acceleration, denoted by $i = S$ and $i = A$, respectively. Diverging from a fixed a_i, UniAda introduces the Adaptive Weighting Scheme for real-time updates to a_i, effectively balancing each objective's contribution.

To generate a universally applicable perturbation that consistently impacts all input images, UniAda finds τ by optimizing the subsequent joint optimization function:

$$\underset{\tau}{\text{argmin}} \sum_{n=1}^{N} O(\mathbf{X}^n_{ori}, \tau) \tag{3.6}$$

The generation of perturbation integrates the joint objective for all image frames within the video. Utilizing a gradient-based technique, UniAda minimizes this joint function, iteratively refining τ.

3.3.3 Adaptive Weighting Scheme

During the multi-objective optimization process $O = \sum_i^C a_i L_i$ (as depicted in Equation 3.5), it is critical to select an appropriate weight for each objective. In our context, they significantly affect the contribution of each individual objective to the perturbation proposal. In the simplest case, one may opt for equally weighted objectives, where each objective is assigned a weight of $a_i = \frac{1}{C}$. In this case, the optimization process may not capture the importance of each objective accurately. Another option involves using computationally expensive grid search methods [139], which systematically explore a predefined set of values for each objective weight and select the combination that maximizes performance. However, it becomes computationally impractical for problems with a large parameter space. In our case, we employ an adaptive method [40] dubbed Adaptive Weighting Scheme (AWS), where a_i can vary at each time step t during the perturbation optimization process. AWS is a complementary component integrated into the multi-objective optimization process, allowing us to determine the optimal value for each a_i at each time step t. This ensures a balance in the contribution of each objective for optimal perturbation searching. Specifically, AWS updates each objective weight based on its gradient to the AWS loss (Equation 3.7) at each step t. AWS operates by assigning larger weights to objectives contributing less (indicating under-training) and smaller weights to those that are overly trained. Through AWS weight updates, both objectives collectively contribute to the perturbation, ensuring effective triggering of errors in both steering and acceleration controls, rather than confining influence to just one.

The AWS loss is formulated to capture differences between the weighted gradient norm for each objective i (reflecting objective i's perturbation pattern's impact on the universal perturbation) and the target norm (representing the desired influence of objective i's perturbation on the universal perturbation). This target influence considers weighted gradient norms of other objectives ($\overline{N}_{\hat{X}}$) and the extent of misbehavior in objective i compared to other objectives $(q_i)^\gamma$. The AWS loss f_i for each i is expressed as follows:

$$f_i = \left| N^i_{\hat{X}} - \overline{N}_{\hat{X}} \times (q_i)^\gamma \right| \tag{3.7}$$

Algorithm 1 Adaptive Weighting Scheme.

Require: $\hat{X}$ - mini-batch images at search time t
Require: a_i - the weight of objective i at time t
Require: L_i^0 - initial loss for objective i
Require: lr_{grad} - hyperparameter, learning rate for AWS update
Require: γ - hyperparameter, strength of rebalance
1: **for** x `in` $\hat{X}$ **do**
2: Compute $L_i(x)$, $\frac{\partial L_i(x)}{\partial x}$ $\forall i$
3: **end for**
4: Compute AWS variables $N^i_{\hat{X}}, \overline{N}_{\hat{X}}, l_i, q_i, f_i^{grad}$ $\forall i$ at time t
5: Conditionally Update $a_i \leftarrow a_i - lr_{grad} \times f_i^{grad}$
6: Normalize $\sum_i a_i = 1$

where f_i consists of four variables, $N^i_{\hat{X}}$, $\overline{N}_{\hat{X}}$, l_i, q_i, with l_i utilized in the computation of q_i. These four variables represent the objective-specific norm, the objective-average norm, the loss ratio that captures the inverse training rate for i, and the relative inverse training rate for i, respectively. The hyperparameter γ controls the strength of rebalance; a larger γ exerts a greater restoring force to pull objectives back to similar training rates. Detailed formulae for each variable are presented below.

1. $N^i_{\hat{X}} = E_{\hat{X}}\left[\|a_i \frac{\partial L_i(x)}{\partial x}\|_2\right]$: the objective-specific norm[1]. $\hat{X} \subseteq \mathbf{X}_{ori} + \tau$ denotes the mini-batch image set at the time t, and $x \in \hat{X}$. $N^i_{\hat{X}}$ denotes the average of weighted gradient norms over all images inside the mini-batch at time t. We treat each image equally when computing the average.

2. $\overline{N}_{\hat{X}} = E_i\left[N^i_{\hat{X}}\right]$: an objective-average norm that averages the objective-specific norms across all objectives.

3. $l_i = \frac{E_{\hat{X}}[L_i(x)]}{L_i^0}$: the loss ratio that captures the inverse training rate for the objective i. A smaller value indicates a higher training rate. Numerator is the batch average loss for objective i at time t. The denominator denotes the initial loss for the objective i, which is the mean loss of the original prediction of i at time zero over all images in the given video. The ratio is used to deal with different loss scales. With the inclusion of this variable, the objective weights are adjusted according to the misbehavior performance of i for the current mini-batch images.

4. $q_i = \frac{l_i}{E_i[l_i]}$: the relative inverse training rate for the objective i, where $E_i[l_i]$ is the mean loss ratio over different objectives. A smaller q_i gives UniAda a hint that the

[1] $\|\cdot\|_2$ denotes Euclidean norm, it is computed after flattening the three-dimensional input matrix to a 1D vector.

objective i is trained too much (i.e., loss L_i decreases too fast) compared to others, in which AWS will decrease its weight.

Algorithm 1 summarizes the AWS process. For each image, AWS computes objective loss $L_i(x)$ and its gradient (Lines 1-3), retaining them for AWS variable computation (Line 4). Line 5 performs an objective weight update if the new weight is positive. The positive weight constraint is grounded in the necessity for $L_i(x)$ and $O(x)$ to be positively correlated, ensuring the minimization of L_i $\forall i$ during $O(x)$ minimization. Finally, the updated weights are normalized to ensure their summation equates to 1 (Line 6).

3.3.4 Generating Adversarial Perturbations

The above two sections constitute two main optimization streams: the perturbation optimization by the joint objective (Equation 3.6) in the universal multi-objective testing, and the objective weight optimization by AWS. Both streams operate in a gradient-based manner and are iteratively updated at each step t. The algorithm details are summarized in Algorithm 2, structured across four main stages: Input, Initialization, Perturbation Searching (comprising two optimization streams), and the Output stage. We elaborate on the details below.

Input: The algorithm requires four key inputs outlined in the **Require** lines. These inputs encompass the input video $\mathbf{X}_{ori}$, the autonomous driving model m responsible for predicting values across multiple car controls, the attack direction d_i corresponding to each objective i, and hyperparameters $hyper$. Table 3.1 explains the role and selected values of each hyperparameter.

Initialization: Lines 1-5 initialize all variables needed. Specifically, Line 1 creates a variable **X** to store all original image frames within the video. Line 2 initializes the universal multi-objective perturbation τ as a tensor of zeros, whose dimension is the same as the input image. Line 3 initializes objective weights a_i uniformly. Line 4 computes the initial loss L_i^0 for objective i, which is used to compute the AWS variable l_i. Lastly, Line 5 sets the training time t as zero.

Perturbation Searching: For each epoch, UniAda shuffles all video images (Line 7) and implements learning rate decay by a predefined schedule (reduced by a factor of 0.8 every 50 epochs, as standard in the DL community [140]) to avoid local optima (Line 8). During each step t of searching, UniAda iterates over mini-batches with batch size bs (Lines 9-10). It subsequently computes single objective loss (Line 11), multi-objective loss (Line 12), and corresponding gradients with respect to each

Table 3.1 UniAda Hyperparameters: describe the meaning of each hyperparameter and the value used in the experiments.

Notation	Description	Value
lr	Perturbation searching learning rate	0.2
lr_{grad}	AWS learning rate	0.005
bs	Batch size	5
β	Sharpness parameter in multi-objective loss	2
$Epochs$	Number of Epochs	250
γ	AWS parameter controlling rebalance strength	10
θ	Perturbation gradient rescale threshold	0.3
ϵ	Maximum perturbation to the image	{2, 5}

image (Line 13). Line 14 involves computation and storage of gradient norms used to determine AWS variable $N^i_{\hat{X}}$. After completing traversal across all mini-batch images in $\hat{X}$, UniAda generates gradient proposals $\frac{\partial O(x)}{\partial x}$ for each image x in the mini-batch, alongside weighted gradient norms $\|a_i \frac{\partial L_i(x)}{\partial x}\|_2$ for each image and objective i. Line 16 calculates average gradient proposals across $\hat{X}$. In instances where gradients are too minimal to impact perturbation τ, inducing sluggish search, UniAda rescales average gradients G by $G = G \times \frac{\theta}{\|G\|_2}$ if $\|G\|_2 < \theta$ and $\|G\|_2 \neq 0$ (Line 17). This amplifies gradient magnitude, elevating its influence during addition to τ. Subsequently, Line 18 updates total perturbation τ via gradient descent, enforcing a maximum perturbation size of ϵ to maintain imperceptibility. Afterward, all image frames within $\mathbf{X}$ undergo updates, subject to processing for perceptual fidelity (Line 19). This entails confining pixel values within the range [0, 255]. Lines 20-22 encapsulate the AWS procedure, as described in Algorithm 1. These steps iteratively unfold until the maximum epoch limit is attained.

Output: Upon reaching the maximum epoch count, UniAda yields a universal multi-objective perturbation τ for the input video.

Algorithm 2 UniAda.

Require: $\mathbf{X}_{ori}$ - input video
Require: m - the targeted autonomous driving model
Require: $d_i \in \{+1, -1\}$ - Direction of attack: accelerate/decelerate, left/right
Require: $hyper - \{lr, lr_{grad}, bs, \beta, Epochs, \gamma, \theta, \epsilon\}$, dictionary of input hyperparameters

$\mathbf{X} = copy(\mathbf{X}_{ori})$
$\tau = zero(\mathbf{X}[0].shape)$
$a_i = \frac{1}{C} \ \forall i \in \{1, ..., C\}$
Compute $L_i^0 \ \forall i$
$t = 0$
for epoch in Epochs **do**
    random.shuffle(**X**)
    Adjust lr, lr_{grad} for each 50 epochs
    for $\hat{X}$ `in` **X do**
        for x `in` $\hat{X}$ **do**
            $L_i(x) = \frac{1}{\beta} \exp\left\{-\frac{1}{\beta}(m_i(x) - m_i(x_{ori})) \times d_i\right\} \ \forall i$
            $O(x) = \sum_{i=1}^{C} a_i L_i(x)$
            Compute $\frac{\partial O(x)}{\partial x}$
            Compute $\|a_i \frac{\partial L_i(x)}{\partial x}\|_2 \ \forall i$
        end for
        $G = \frac{1}{len(\hat{X})} \sum_{x \in \hat{X}} \frac{\partial O(x)}{\partial x}$
        Conditionally Rescale G
        Update $\tau \leftarrow Clip_{\epsilon}(\tau - lr * G)$
        Update $\mathbf{X} \leftarrow Process(\mathbf{X}_{ori} + \tau)$
        Compute $N^i_{\hat{X}}, \overline{N}_{\hat{X}}, l_i, q_i, f_i^{grad} \ \forall i$
        Conditionally Update $a_i \leftarrow a_i - lr_{grad} \times f_i^{grad}$
        Renormalize $\sum_i a_i = 1$
        $t = t + 1$
    end for
end for
return τ

3.4 Experimental Setup

In this section, we introduce the studied research questions, datasets and models, baselines, evaluation metrics, and hyperparameters used in the experiments.

3.4.1 Research Questions

RQ1. UniAda Effectiveness: *How effective is UniAda in generating adversarial perturbations?*

To evaluate the effectiveness of UniAda in generating perturbations that trigger errors in multiple controls, we compare its performance on 14 driving videos (7 simulated and 7 real-world videos) with four existing methods (DeepManeuver, Perturbation Attack, DeepBillboard, FGSM) on two evaluation metrics (mean error and success rate).

RQ2. Adaptive Weighting Scheme Effectiveness: *How can the Adaptive Weighting Scheme assist the attack performance?*

To assess the effectiveness of AWS, we introduce a variant of UniAda dubbed UniEqual, wherein objectives are assigned equal weights during the perturbation search process. By comparing UniEqual with UniAda, we can observe the exact impact of AWS. Furthermore, we graphically depict the trajectory of each objective weight to provide insights into the operation of AWS. We also conduct the statistical analysis (t-test) to show the significance of the improvement.

RQ3. Multi-objective Attack Effectiveness: *Does the multi-objective adversarial attack improve the attack effectiveness for all objectives simultaneously compared to the single-objective attack?*

Attacking multiple objectives together may yield a better result for each objective than single-objective attacks, since attacking diverse objectives can offer shared insights sometimes, yielding a more comprehensive and effective perturbation pattern. To answer this question, we compare the effectiveness of the adversarial perturbation generated by the multi-objective loss (Equation 3.5) with that by the single objective loss (Equation 3.4).

3.4.2 Datasets & Models

We evaluate UniAda on three widely used multi-output autonomous driving models with 14 driving videos. The details are elaborated below.

Table 3.2 Descriptions for the selected videos

Videos (No. of Imgs)	Description
Carla Pedestrian (18)	Stop for a pedestrian, clear sunset
Carla White Car (36)	Stop for the front white car, wet noon
Carla Black Car (22)	Stop for the front black car, clear noon
Carla Gray Car (42)	Stop for the front gray car, clear sunset
Carla Blue Car (43)	Driving Straight, following a blue car, clear sunset
Carla Red Light (66)	Stop for the front red light, wet noon
Carla Light-blue Car (21)	Stop for the front light-blue car, clear noon
Dave Curve1 (34)	Turning left at crossroad, clear noon
Dave Straight1 (54)	Driving straight, urban two-way road, clear noon
Udacity Straight1 (21)	Stop for a white SUV, urban one-way road, clear noon
Udacity Straight2 (22)	Driving Straight, urban one-way road, clear noon
Udacity Curve1 (15)	Turning left, urban two-way road, dusk
Kitti Curve1 (21)	Turning Right, urban one-way road, clear noon
Kitti Straight1 (21)	Driving Straight, urban one-way road, clear noon

Datasets: Our experimentation is grounded in the use of 14 driving videos in total, with 7 simulated videos and 7 real-world videos. For the simulated videos, we extracted all 7 videos from the Carla100 dataset [25], which contains realistic simulated driving data (e.g., RGB images, steering, and speed-related control data) captured by the central camera with 100 hours of driving from an urban town of the Carla simulator (version 0.8.4). For the real-world videos, we extracted: (1) Two videos from Dave testing dataset [42] which contains 45,568 real-world driving images to test the NVIDIA Dave model; (2) Three videos from Udacity self-driving car challenge dataset [26] that contains 101,396 real-world driving images captured by a dashboard-mounted camera of a human-driven car; (3) Two videos from Kitti dataset [41] which contains 14,999 real-world driving images from six different scenes captured from a VW Passat station wagon equipped with four cameras.

Our manual selection of videos captures various scenarios under urban traffic, such as different weather conditions and different driving maneuvers. Further details regarding these 14 videos are summarized in Table 3.2.

Autonomous Driving Models: In the assessment, we select three end-to-end autonomous driving models for testing: CILRS, CILR, and MotionTransformer. Table 3.3 lists the dataset used for training and validation and the performance of models. Following common practice [4, 15] in autonomous driving research, we report Mean Squared Error (MSE) between actual and predicted values as a measure of model accuracy. For consistency, we compute MSE on steering in radians and acceleration in [0,1] for all models on their provided validation sets.

CILRS and CILR are conditional imitation learning models, proposed by Codevilla *et al.* [25]. CILR uses the ResNet perception module as the backbone to extract information from RGB inputs. CILRS extends CILR architecture with an additional speed prediction head to incorporate speed-related features into the representation. They achieve outstanding performance in Carla's simulated urban traffic scenarios. Both models are trained using 10 hours of expert demonstrations and validated on a 2-hour subset (extracted from the Carla100 dataset). For both models, we use publicly available pre-trained weights [2]. Readers can refer to the original work [25] for training details (e.g., training objectives, hyperparameter settings). On the provided validation set, both models achieve 0.0002 MSE in steering and approximately 0.03 MSE in acceleration.

MotionTransformer (MT) [141] utilizes RGB and optical flow images to learn both position and motion information. It employs two ResNet backbones for feature extraction from RGB and optical flow images, followed by a transformer encoder to distill knowledge from both. We use publicly available pre-trained weights provided by the original work [3]. During their training process (please refer to the original work [141] for training details), the first 90% of samples from the Udacity self-driving car challenge dataset are used for training, and the remaining 10% is used for validation. On the validation set, MT achieves 0.002 MSE in steering and 0.0079 in acceleration, surpassing the widely-used steering angle prediction model, NVIDIA's DAVE2 [1], which only achieves 0.0449 MSE in steering based on our experiments.

For the experimentation, only the RGB input undergoes perturbation, while other inputs (if there exists) remain unchanged. To maintain consistency and simplicity, we convert the output steering values from radians into degrees (°) and acceleration values into speed values (km/h) according to the data documentation.

[2] https://github.com/felipecode/coiltraine

[3] https://github.com/chingisooinar/AI_self-driving-car

Table 3.3 Model Performance (MSE) on the provided validation set.

Train & Val set	Model	Performance (MSE)	
		Steering	Acceleration
Carla100	CILR	0.0002	0.0382
Carla100	CILRS	0.0002	0.0343
Udacity	MT	0.0020	0.0079

3.4.3 Baselines

We test five baselines to evaluate the effectiveness of UniAda. To ensure a fair comparison, for all methods, we set a uniform maximum perturbation, we set the attack surface to the whole image, and we perform targeted adversarial attacks. We use the best hyperparameter values reported in their paper for experiments.

Fast Gradient Sign Method (FGSM): Proposed by Goodfellow *et al.* [30], FGSM is a one-step gradient-based attack approach to craft image-specific adversarial perturbations. To facilitate a meaningful comparison with UniAda, we extend FGSM to perform targeted attacks on multiple car controls by employing an equal-weighted multi-objective loss (same loss as UniEqual) as the optimization criterion during the perturbation search.

DeepBillboard (DB): Introduced by Zhou *et al.* [27], DeepBillboard represents an iterative state-of-the-art attack method. It performs targeted attacks (intentionally misleads the model to steer left or right) and consistently misleads the model's steering predictions. The adversarial perturbation pattern generation process is intrinsically linked to the constraint applied to the steering angle. Due to this inherent constraint, unlike FGSM, it is challenging to extend DeepBillboard to multiple objectives. Consequently, we present the steering angle results exclusively.

DeepManeuver (DM) [36] is a state-of-the-art gradient-based adversarial test generation approach for autonomous vehicles. It generates a perturbation patch on the predefined attack surface, consistently misleading the model's steering prediction into the targeted direction (i.e., turn left or right). Similarly, we exclusively report steering results due to the design of its algorithm.

Perturbation Attack (PA) [43] assesses the impact of image-specific perturbation attacks and patch attacks on the 3D object detector. The former applies the attack to the entire image, while the latter achieves the attack by applying a small patch to the input image. To enable a meaningful comparison with UniAda, we utilize

the perturbation attack and an equal-weighted multi-objective loss (the same loss as UniEqual, enabling the specifications of attack directions) as the optimization criterion to induce misbehavior in car controls instead of 3D bounding boxes.

UniEqual: A variant of UniAda, sharing an identical universal multi-objective attack algorithm. However, UniEqual assigns equal weights to different objectives, omitting the use of AWS. Similar to UniAda, it is an iterative attack method designed to generate a universal multi-objective perturbation.

3.4.4 Evaluation Metrics

In this section, we introduce the evaluation metrics utilized in our experimentation. To assess the attack effectiveness of the generated examples, we adhere to established practices [50, 27] and employ two widely-used metrics for the offline evaluation of ADS models: Mean Error (ME) [101] and Success Rate (SR) [50], as defined below for each objective i:

$$ME_i = \frac{1}{N}\sum_{n=1}^{N}[d_i \times (m_i(\mathbf{X}_{ori}^n + \tau) - m_i(\mathbf{X}_{ori}^n))] \tag{3.8}$$

$$SR_i = \frac{1}{N}\sum_{n=1}^{N} I(d_i \times (m_i(\mathbf{X}_{ori}^n + \tau) - m_i(\mathbf{X}_{ori}^n)) > \delta_i) \tag{3.9}$$

The variables mentioned above are detailed in Section 3.1. I is an indicator function that returns 0 or 1. Notably, the evaluation metrics are designed to capture the power of the attack. Larger metric values signify more effective attacks. In order to punish incorrect attack directions (i.e., situations when the attacker wishes the car to turn right but the attack algorithm misleads the car into turning left), the use of sign-invariant operators such as absolute or squared values in the metrics is avoided.

Mean error ME indicates the average strength of our attack on the input video. Success rate SR provides a hierarchy of performance by segmenting results based on different thresholds. More specifically, the success rate measures the proportion of image frames in the video that have been successfully attacked. An attack is deemed successful if the discrepancy between the prediction made on the generated adversarial image and the original prediction surpasses a predetermined threshold. Thresholds for steering and acceleration are denoted as δ_S and δ_A, respectively. By testing various threshold values, we gain hierarchical insights into how the universal adversarial perturbation impacts a sequence of model predictions.

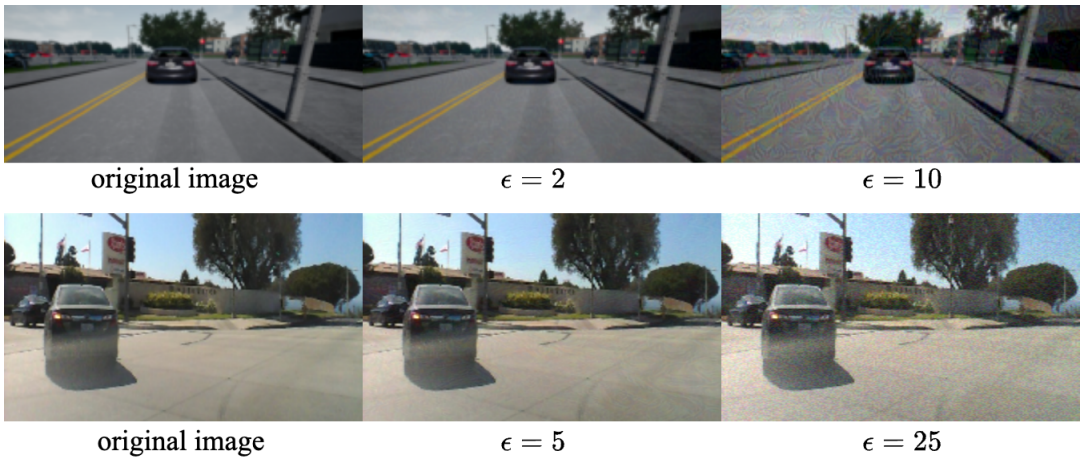


Fig. 3.3 Adversarial images with different ϵ values.

3.4.5 Hyperparameters

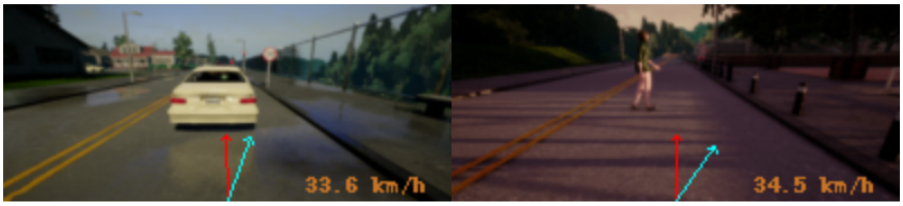
For the baseline methods, we use the best hyperparameters reported in their paper for the experimentation. Hyperparameters for UniAda are outlined and explained in Table 3.1.

To ensure the generated perturbations remain imperceptible to humans, we set the maximum perturbation to ϵ=2 for simulated videos and ϵ=5 for real-world videos. This value is chosen to strike a balance between inducing misbehavior and maintaining the visual similarity between the original and perturbed images. In Fig. 3.3, we illustrate examples of adversarial images produced with perturbations of different ϵ values. Notably, for simulated images with ϵ=2 and real-world images with ϵ=5, the perturbed image closely resembles the original. For higher ϵ values, the distinction between an adversarial image and its original counterpart becomes more evident.

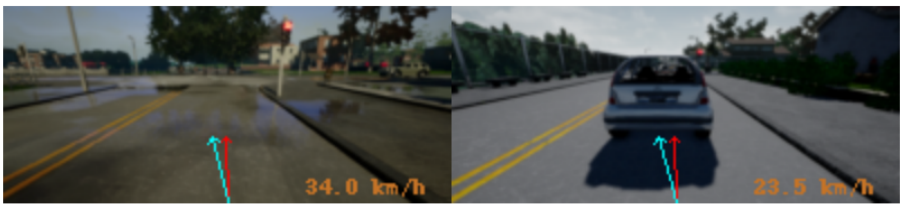
Reproducibility. The trained model, framework, and data are available at https://github.com/UniAdaRepo/UniAda/.

3.5 Results and Analyses

In this section, we present our results and analyses to address the research questions. For all methods, the targeted attack directions are set to d_S=1 (steering to the right) and d_A=1 (acceleration). Positive/Negative error values indicate alignment/misalignment with the specified attack directions. All experiments are repeated five times, and the average results are reported.



(a) Carla White Car, right (b) Carla Pedestrian, right



(c) Carla Red Light, left (d) Carla Light-blue Car, left

(e) Dave Curve1, right (f) Kitti Curve1, right

(g) Dave Straight1, left (h) Udacity Straight1, left

Fig. 3.4 Adversarial Images generated by UniAda: The first and second (third and fourth) rows are adversarial images generated from simulated (real-world) videos. Blue/Red arrows indicate the adversarial/original steering angle predicted from the image. The bottom right shows the acceleration error.

3.5.1 Visualization of Generated Adversarial Images

We provide visualizations of example adversarial images generated by UniAda in Fig. 3.4 for both simulated and real-world data. The top/bottom row displays adversarial images generated from a simulated/real-world dataset. Red/blue arrows represent the original/adversarial steering angle prediction. The bottom-right corner of each image indicates the acceleration error.

3.5.2 RQ1. UniAda Effectiveness.

We evaluate the effectiveness of UniAda in comparison to four baselines: DeepManeuver, Perturbation Attack, DeepBillboard, and FGSM, using two evaluation metrics across 14 videos and 3 ADSs.

Mean Error Results: Results for simulated and real-world data are presented in Table 3.4 and Table 3.5. UniAda outperforms all four baselines in both steering angle and acceleration objectives in the average results for both simulated and real-world data. For simulated videos, under CILRS, UniAda causes an average steering error of approximately 29.2°, which is 6.6°, 9.1°, 12.6°, and 28.2° greater than DM, DB, PA, and FGSM, respectively. For acceleration, UniAda leads to an acceleration error of around 22.3 km/h, surpassing PA and FGSM by 3.4 km/h and 11.5 km/h, respectively. For CILR, UniAda maintains its leading position for both objectives, causing an average steering error of 25.6°, with a maximum error of 43.4° under "Red Light". For acceleration, it reaches an average error of 19.6 km/h. Among the four baselines, for both models on average, DM demonstrates the best performance, while FGSM performs the worst. DB and PA secure middle positions. Delving into the performance of individual videos, we found that under the CILR "Light-blue Car" video, UniAda is slightly outperformed by DB in ME_S by around 3.4°. Similarly, in CILRS "Pedestrian", UniAda is slightly outperformed by DM by 0.6°. However, DB and DM can only target one objective at a time, making them less functional compared to UniAda. Furthermore, when considering other videos and the average result, UniAda consistently outperforms them by a significant margin.

For real-world videos, UniAda reaches the best performance on average, with 3.54° in ME_S and 11.0 km/h in ME_A, which is about triple ME_S of DB. Only in one case ("Kitti Curve1") DB slightly outperforms UniAda by 0.24°. On average, PA performs the best among four baselines, but is 9.47 km/h and 0.85° smaller than UniAda. DM yields the worst performance, resulting in 0.61° in ME_S, which may be attributed to the fact that DM is designed for online testing within simulators.

Table 3.4 Mean Error Result Table: mean error results for 7 simulated videos of all six techniques, tested with CILRS and CILR models. The last line indicates the average result over 7 videos. Best results are highlighted in bold. We approximate the floating steering angle to degrees ° in ME_S and floating acceleration to km/h in ME_A for a more straightforward attack effect. - denotes unavailable results.

Videos	Metrics	CILRS					
		UniAda	**DM**	**DB**	**PA**	**FGSM**	**UniEqual**
Carla	ME_S	39.9	**40.5**	32.0	32.6	1.40	39.5
Pedestrian	ME_A	22.9	-	-	16.1	15.6	**23.2**
Carla	ME_S	**25.6**	18.1	20.5	16.1	1.40	24.6
White Car	ME_A	**22.0**	-	-	20.7	15.6	21.1
Carla	ME_S	**18.6**	6.25	5.47	13.5	0.70	9.63
Black Car	ME_A	23.8	-	-	16.8	8.70	**23.9**
Carla	ME_S	**23.7**	11.8	5.88	12.0	0.70	7.08
Gray Car	ME_A	**17.4**	-	-	13.9	6.90	14.7
Carla	ME_S	**28.6**	24.8	26.1	14.5	1.40	22.3
Blue Car	ME_A	**22.6**	-	-	21.9	8.74	22.5
Carla	ME_S	**41.9**	38.9	35.4	21.2	0.70	41.2
Red Light	ME_A	**26.3**	-	-	23.1	11.0	26.2
Carla	ME_S	**25.9**	17.5	15.0	6.37	0.70	11.2
Light-blue Car	ME_A	21.0	-	-	19.9	8.74	**22.0**
Average	ME_S	**29.2**	22.6	20.1	16.6	1.00	22.2
	ME_A	**22.3**	-	-	18.9	10.8	21.9
Videos	**Metrics**	**CILR**					
		UniAda	**DM**	**DB**	**PA**	**FGSM**	**UniEqual**
Carla	ME_S	**30.1**	23.9	12.7	22.5	0.20	30.0
Pedestrian	ME_A	**13.3**	-	-	11.6	9.70	13.2
Carla	ME_S	**19.6**	10.4	9.79	11.1	0.90	19.5
White Car	ME_A	**15.4**	-	-	14.6	12.9	14.8
Carla	ME_S	**19.6**	12.3	6.81	9.51	1.20	13.8
Black Car	ME_A	**25.4**	-	-	18.6	6.00	21.3
Carla	ME_S	**14.8**	2.21	12.0	11.8	1.20	13.3
Gray Car	ME_A	**18.2**	-	-	16.5	5.06	18.1
Carla	ME_S	**31.2**	18.5	9.68	15.1	0.10	30.7
Blue Car	ME_A	**24.3**	-	-	21.2	8.28	23.8
Carla	ME_S	**43.4**	31.9	36.5	17.9	0.60	42.7
Red Light	ME_A	26.3	-	-	21.8	9.20	**26.4**
Carla	ME_S	20.4	19.6	**23.8**	5.25	0.80	7.85
Light-blue Car	ME_A	**14.0**	-	-	0.91	0.92	4.16
Average	ME_S	**25.6**	17.0	15.9	13.3	0.71	22.6
	ME_A	**19.6**	-	-	15.0	7.44	17.4

Table 3.5 Mean Error Result for 7 real-world driving videos of all six techniques with the MotionTransformer model under test. Avg. denotes the average results over 7 videos.

Videos	**Metrics**	**MotionTransformer**					
		UniAda	**DM**	**DB**	**PA**	**FGSM**	**UniEqual**
Dave Curve1	ME_S	**7.25**	1.54	1.88	5.69	0.94	5.73
	ME_A	**14.3**	-	-	5.21	2.54	3.97
Dave Straight1	ME_S	**1.01**	-0.01	0.25	0.30	-0.08	0.63
	ME_A	**4.68**	-	-	-5.35	-7.55	-2.33
Udacity Straight1	ME_S	**8.15**	7.47	7.31	8.11	4.51	7.99
	ME_A	**16.5**	-	-	10.1	3.55	6.18
Udacity Straight2	ME_S	**2.75**	-0.25	-0.01	0.73	0.14	1.57
	ME_A	**10.3**	-	-	-4.65	-2.06	1.29
Udacity Curve1	ME_S	**1.97**	-3.09	-2.18	1.07	0.83	1.94
	ME_A	**12.2**	-	-	3.00	1.99	-2.35
Kitti Curve1	ME_S	2.90	2.76	**3.14**	2.90	0.84	3.10
	ME_A	**7.20**	-	-	1.41	0.34	-0.59
Kitti Straight1	ME_S	**0.74**	-4.16	-2.45	0.09	0.54	0.63
	ME_A	**11.7**	-	-	1.01	-1.65	1.75
Avg.	ME_S	**3.54**	0.61	1.13	2.69	1.10	3.08
	ME_A	**11.0**	-	-	1.53	-0.41	1.13

Targeted Attack Effectiveness: From Table 3.4, all methods demonstrate positive error values, indicating their effectiveness in misleading CILRS and CILR in the intended direction. However, when attacking the MT model (Table 3.5), all methods except UniAda occasionally generate adversarial samples that mislead the model into unintended directions, thereby failing to accomplish targeted attacks. Additionally, all techniques achieve a relatively low error compared to attacking CILRS and CILR. The weak performance in attacking the MT model may be attributed to the fact that, according to our settings, only the RGB input undergoes perturbations, whereas the MT model extracts features from both RGB and optical flow images. Out of 7 videos, FGSM induces MT to decelerate in 3 cases, and 2 cases for PA (i.e., negative ME_A, which contradicts the intended acceleration target). In teams of steering angle, FGSM incorrectly induces the MT to turn left instead of right as instructed in one case, while PA, DB, and DM do so in zero, three, and four instances, respectively. Notably, FGSM exhibits more successful targeted attacks than state-of-the-art DB and DM, while also achieving a larger average ME_S than DM does. One possible reason for DB could be attributed to its algorithm design, which updates the perturbation based on the absolute steering error, neglecting its sign. Meanwhile, DM is originally designed for online testing within a simulated environment, thereby lacking the ability to generalize effectively to real-world data. Moreover, both DM and DB apply a single, image-agnostic perturbation across the entire video, whereas FGSM and PA generate unique perturbations to each image (image-specific). This may potentially limit DB and DM's efficacy in targeted attacks on models with complex architectures.

Success Rate Results: The success rate, indicating the percentage of adversarial images successfully exhibiting misleading behavior under a specified threshold, is assessed for different bounds. In Table 3.6, we select four different bounds for each objective to assess the hierarchy of performance, with δ_S=3.5°, 14°, 21°, 28° for the steering angle and δ_A=4.6, 13.8, 23.0, 32.2 km/h for acceleration.

UniAda consistently outperforms four baseline methods across all thresholds and all models, except for the case when δ_S=28° under MT model. For example, UniAda successfully attacked CILRS model to produce a steering prediction error of at least 28° for 45.4% of tested images, compared to DM's 38.9%, DB's 34.3%, PA's 15.1%, and FGSM's 0% at the same threshold. For MT model, UniAda can trigger errors on much more images than baselines for acceleration, with at most 43.5% more images than the second best (i.e., PA). FGSM fails to produce a successful attack under nearly half of the tested thresholds.

Table 3.6 Mean Success Rate Result average over all testing videos for three ADSs: CILRS and CILR results are averaged over 7 Carla100 videos. MT results are averaged over 7 real-world driving videos. We tested thresholds δ_S=3.5, 14, 21, 28 in degrees and δ_A=4.6, 13.8, 23.0, 32.2 in km/h.

Methods		CILRS				CILR				MT			
	δ_S	3.5	14	21	28	3.5	14	21	28	3.5	14	21	28
	δ_A	4.6	13.8	23.0	32.2	4.6	13.8	23.0	32.2	4.6	13.8	23.0	32.2
UniAda	SR_S	**96.3**	**89.5**	**76.5**	**45.4**	**90.5**	**75.3**	**62.2**	**45.1**	**33.5**	**12.3**	**8.34**	5.34
	SR_A	**74.7**	**67.4**	**62.3**	**49.2**	**67.8**	**61.1**	**55.7**	**43.9**	**83.7**	**31.6**	**2.72**	0.00
DM	SR_S	87.6	67.2	52.5	38.9	83.4	57.3	33.5	19.6	27.9	11.2	8.32	5.34
	SR_A	-	-	-	-	-	-	-	-	-	-	-	-
DB	SR_S	83.3	48.8	45.6	34.3	66.1	41.7	29.2	20.2	28.2	11.2	8.32	6.02
	SR_A	-	-	-	-	-	-	-	-	-	-	-	-
PA	SR_S	86.2	54.6	24.3	15.1	82.0	38.6	18.6	11.0	30.7	11.9	7.68	5.38
	SR_A	67.4	60.8	52.3	41.2	55.2	48.0	41.7	28.7	40.2	9.56	0.68	0.00
FGSM	SR_S	3.99	0.00	0.00	0.00	1.32	0.00	0.00	0.00	9.89	3.67	0.79	0.00
	SR_A	43.6	35.4	28.8	18.3	37.2	27.5	21.6	11.4	16.8	0.00	0.00	0.00
UniEqual	SR_S	83.4	57.9	48.0	37.6	84.2	57.3	47.1	38.8	32.0	11.6	8.06	6.02
	SR_A	73.7	65.9	60.1	47.6	59.4	52.6	47.3	36.3	28.3	5.85	0.00	0.00

Result 1: UniAda consistently outperforms all four baseline methods in Mean Error on average for all ADS models. For the Success Rate, UniAda demonstrates the best performance under 23 out of 24 cases on average. Compared with the state-of-the-art technique, DM, UniAda achieves a mean steering error improvement of 6.6° in CILRS, 8.6° in CILR, and 2.93° in the MT model.

3.5.3 RQ2. Adaptive Weighting Scheme Effectiveness.

To assess the effectiveness of the Adaptive Weighting Scheme (AWS), we introduce a variant of UniAda dubbed UniEqual, which utilizes the same universal multi-objective attack algorithm but employs equal-weighted objectives during the perturbation search process (i.e., UniEqual omits the use of AWS).

Observing the Mean Error results in Table 3.4 and Table 3.5, it is evident that UniAda consistently outperforms UniEqual in both steering and acceleration objectives on average across all ADS models. This indicates that AWS effectively enhances the attack performance in both objectives simultaneously. While in certain videos, UniEqual might achieve a slightly higher attack error in one objective, it becomes apparent that it is due to the lack of focus on the other objective. For instance, in the CILRS model videos “Black Car” and “Light-blue Car”, UniEqual produces 23.9 km/h and 22.0 km/h acceleration error, only 0.1 km/h and 1.0 km/h higher than UniAda,

Table 3.7 P-value results of Mean Error under CILRS, CILR and MT: results are shown in bold if they have a p-value ≤ 0.05, The positive/negative/equal signs indicate that UniAda has a better/worse/equal performance versus UniEqual.

Videos	**CILRS**		**CILR**	
	ME_S	ME_A	ME_S	ME_A
Carla Pedestrian	7.61E-01(+)	1.67E-01(-)	8.88E-01(+)	**3.95E-03(+)**
Carla White Car	**3.49E-02(+)**	**3.19E-02(+)**	8.05E-01(+)	**2.64E-02(+)**
Carla Black Car	**5.93E-03(+)**	7.85E-01(-)	**4.39E-02(+)**	**1.53E-02(+)**
Carla Gray Car	**6.28E-07(+)**	**7.48E-07(+)**	**7.40E-05(+)**	9.25E-01(+)
Carla Blue Car	**3.58E-02(+)**	5.38E-01(+)	**4.99E-05(+)**	**1.73E-08(+)**
Carla Red Light	**4.37E-02(+)**	8.62E-01(+)	**2.97E-03(+)**	1.50E-01(-)
Carla Light-blue Car	**4.28E-07(+)**	3.81E-01(-)	**2.63E-04(+)**	**1.05E-05(+)**
Average	**2.72E-06(+)**	2.63E-01(+)	**8.89E-04(+)**	**1.43E-04(+)**

(a) P-value results of CILRS and CILR

Videos	**MotionTransformer**	
	ME_S	ME_A
Dave Curve1	**2.58E-08(+)**	**1.18E-09(+)**
Dave Straight1	**1.20E-02(+)**	**1.71E-03(+)**
Udacity Straight1	**1.28E-02(+)**	**1.27E-10(+)**
Udacity Straight2	**2.29E-03(+)**	**1.43E-05(+)**
Udacity Curve1	6.03E-01(+)	**4.08E-15(+)**
Kitti Curve1	1.75E-01(-)	**3.00E-04(+)**
Kitti Straight1	**4.75E-04(+)**	**3.76E-12(+)**
Average	**2.71E-05(+)**	**1.08E-09(+)**

(b) P-value results of MT

Table 3.8 P-value results of Success Rate under CILRS, CILR, and MT. 'NA' denotes unavailable results due to the same performance of UniAda and UniEqual.

δ_S (degrees)	3.5	14	21	28
δ_A (km/h)	4.6	13.8	23.0	32.2
CILRS	**2.64E-04(+)**	**1.39E-04(+)**	**7.44E-05(+)**	**1.15E-03(+)**
	3.93E-02(+)	**4.09E-02(+)**	**2.32E-02(+)**	9.92E-02(+)
CILR	5.43E-02(+)	**5.37E-03(+)**	**2.98E-02(+)**	**4.59E-02(+)**
	1.37E-03(+)	**2.17E-03(+)**	**1.35E-03(+)**	**2.29E-02(+)**
MT	3.40E-01(+)	**9.23E-05(+)**	**5.56E-09(+)**	7.32E-02(-)
	3.72E-08(+)	**7.01E-08(+)**	**2.97E-11(+)**	NA(=)

respectively. However, UniAda showcases a steering error of 18.6° and 25.9°, which is significantly higher than UniEqual, with 8.97° and 14.7°, respectively.

For real-world driving videos, UniAda consistently outperforms UniEqual in all seven videos, with an average improvement of 9.87 km/h in acceleration error and 0.46° in steering error. We can see that AWS maintains the attack strength on steering objective while significantly improves the attack power on acceleration. Moreover, it is noteworthy that in 3 out of 7 cases, UniEqual performs attack in an incorrect direction for the acceleration objective (i.e., misalignment with the specified direction). In contrast, AWS optimizes UniAda to successfully attack all tested videos in the correct direction as specified. This is attributed to the AWS feature of balancing the training rate of both objectives by adapting weights, which is to ensure both objectives are thoroughly explored and collectively contribute to the perturbation that significantly affects both objectives simultaneously.

A similar conclusion applies to the Success Rate results. As displayed in Table 3.6, UniAda outperforms UniEqual in all models across all tested thresholds, except for one instance: δ_S=28° under the MT model. In this case, perturbation generated by UniEqual can mislead slightly more images than UniAda with only 0.68% improvement. Nevertheless, with the same perturbation, UniAda can mislead 55.4%, 25.8%, 2.72% more images than UniEqual at δ_A=4.6 km/h, 13.8 km/h, 23.0 km/h, respectively.

Statistical Significance: We have conducted two-sample t-tests [142] to assess whether the performance differences between UniAda and UniEqual are statistically significant. Specifically, we treat all five runs of results as a sample group, with the null hypothesis stating that the means of two populations are the same. Each evaluation metric is examined for each driving model for both real-world and simulated driving data. Table 3.7 and Table 3.8 demonstrate the p-value results for Mean Error and Success Rate, respectively. Individual video results offer a detailed performance comparison for specific scenarios. To examine the statistical significance of the overall performance difference under the simulated/real-world driving environment, we also present p-values for the average results (for each run, we compute the average results of 7 videos, and then perform t-tests with 5 runs as a group). Results are shown in bold if they have a p-value ≤ 0.05, the positive/negative/equal signs indicate that UniAda has a better/worse/equal performance versus UniEqual. For Mean Error p-value results, there are 30 out of 42 cases (exclude average results) UniAda statistically significantly outperforms UniEqual. Out of these, 8 cases come from CILRS, 10 cases for CILR, and 12 cases from MT. Moreover, we note that for the videos that UniEqual outperforms

UniAda, the results are not significant (e.g., “Black Car”, “Light-blue Car”). For Success Rate p-value results, in 19 out of 23 cases, UniAda is significantly better than UniEqual.

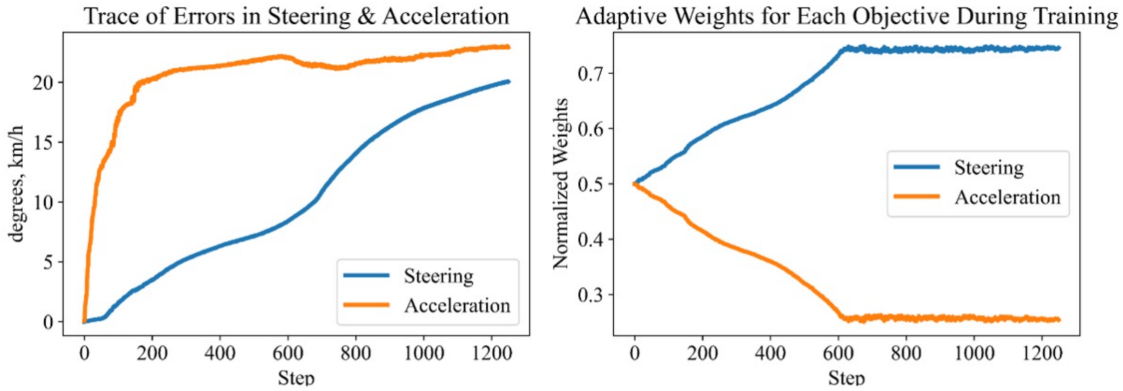


Fig. 3.5 Traces of mean error (left) and AWS normalized weights (right) by UniAda attack in CILRS “Black Car”.

To illustrate how AWS functions, we demonstrate an example plot (CILRS “Black Car”, Fig. 3.5) of error and weight traces against the perturbation searching process. Each objective weight begins at 0.5 with zero errors. During the initial stages, the attack error in acceleration experiences a substantial improvement, whereas the steering perturbation triggers minimal errors. This discrepancy may be attributed to the higher training rate allocated to acceleration. Consequently, the AWS assigns more weight to the steering objective, driving up the steering error while stabilizing the acceleration error after around 200 steps. The objective weights reach a steady state after approximately 600 steps. At the later stage, although the weights are mainly allocated to steering, the acceleration error still shows a further increase, which implies the perturbation pattern may contain shared information that also triggers errors in acceleration.

Result 2: AWS assists UniAda to improve the performance in both objectives simultaneously in most cases. On average, AWS improves ME_S by 7.0°, 3.0°, 0.46°, and ME_A by 0.4 km/h, 2.2 km/h, 9.87 km/h, for CILRS, CILR, and MT, respectively.

3.5.4 RQ3. Multi-objective Attack Effectiveness.

To assess the effectiveness of the multi-objective attack, we conduct experiments involving two single-objective attacks and compare their results with UniAda for CILRS and MT models. The single-objective attacks utilize the same algorithm as UniAda, differing solely in the loss function employed. We report the average results over the corresponding testing videos, summarized in Table 3.9, with attack directions set at d_S=1 and d_A=1.

Table 3.9 Ablation study on CILRS and MT models, averaged over testing videos

	CILRS			MT		
	L_{Steer}	L_{Acc}	UniAda	L_{Steer}	L_{Acc}	UniAda
ME_S	23.7	-	29.2	3.26	-	3.54
ME_A	-	19.5	23.0	-	9.82	11.0

On average, UniAda consistently outperforms both single-objective attacks. Specifically, in terms of steering, UniAda induces a steering angle misdirection of 5.5° more than the single-objective attack focused on steering (L_{Steer}). Similarly, for acceleration, UniAda generates an acceleration error around 3.5 km/h higher compared to the single-objective acceleration attack (L_{Acc}). Similar results for MT model, where the single-objective attack is 0.28° and 1.18 km/h less than that of UniAda. This outcome suggests that the perturbation patterns sought for steering and acceleration objectives complement each other, showcasing the advantage of multi-objective attacks.

Result 3: Compared to single-objective attacks, the multi-objective attack boosts the performance for both objectives simultaneously for both simulated and real-world videos.

3.6 Discussion

3.6.1 Transferability of Adversarial Attacks

In this section, we analyze the transferability of the generated adversarial perturbation on different autonomous driving models. Specifically, we explore the attack effectiveness of the perturbation generated with CILRS on the CILR model for each

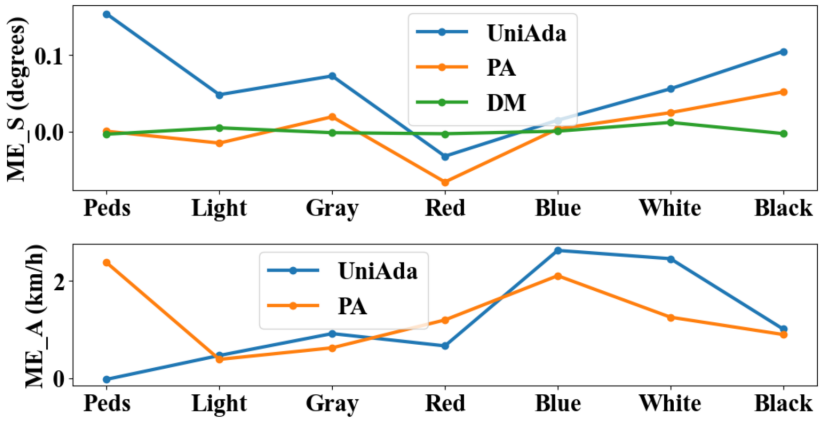


Fig. 3.6 Transferability of Adversarial attacks: Mean Error results in Steering (top) and Acceleration (bottom) of UniAda, DM, PA. Note that DM results for ME_A is not available.

video in Carla100 dataset. We compare the Mean Error results of UniAda with its two closest competitors, DM and PA, shown in Figure 3.6.

As shown in the plot, for ME_S, the perturbation generated by UniAda with the CILRS model under test can cause the highest error when attacking the CILR model in most cases. Only in the video "Red Light", DM produces a better performance. For ME_A, UniAda outperforms PA in 5 out of 7 cases. Only in "Pedestrian", PA outperforms UniAda by a relatively large margin. However, we can see that all three techniques exhibit weak transferability, a phenomenon frequently observed under the white-box setting owing to the attack's specificity to the internal structure of the model [143]. Possible directions to address this issue include using advanced gradient calculation [105], or applying various input transformations [47]. We leave the improvement of transferable attacks for further work.

3.6.2 Limitations

UniAda is a gradient-based approach that requires white-box access, leading to less generalizability on DNNs with restricted access or knowledge. Moreover, our findings on real-world videos is limited by one victim DNN, MotionTransformer. The choice of MotionTransformer is due to its complex architecture (i.e., it incorporates RGB and optical flow images to learn both position and motion information) and its competitive performance (e.g., surpassing the CNN-based DAVE2 model). UniAda also operates under the assumption that the attack direction for each control objective is known in advance. This might lead to ineffectiveness in more complex or automated attack

scenarios, as pre-defined attack directions can be sub-optimal. To extend to adaptive attack direction inference, the future work can explore training a light-weight attack policy to predict conceptually attack directions from real-time input features of the surrounding environment, and integrate this light-weight policy with UniAda to realize a fully automated attack process with adaptive and optimal attack directions. Another consideration is the coupling effect of multi-objective loss optimization. In our case, we have studied and discovered that multi-objective optimization is more effective than single-objective optimization. This may not hold for every scenario and other objectives (other than steering and acceleration). Our future work aims to explore gradient surgery [144] to solve this issue when there are conflicting gradients.

3.6.3 Threats to Validity

Internal Validity: The threats to internal validity can be attributed to the implementation quality of baselines and reproducibility of our method. To mitigate this, we open-source our implementation to facilitate checking and reproducibility.

External Validity: The first threat to external validity to our experimental conclusions is our selected datasets (i.e., 7 simulated videos from Carla100 dataset, 7 real-world videos from Dave, Kitti, and Udacity datasets) and DNN models (i.e., CILRS, CILR, MT). We tried to alleviate this threat as follows: (1) the selected datasets contain both simulated and real-world driving scenarios. They are also widely used in the previous research [27, 28, 25]; (2) the three autonomous driving models have achieved competitive performance in the field. They are constructed with different architectures and different numbers of layers. For example, CILR and CILRS use ResNet only to learn RGB data information, while MT is composed of more complex architecture that incorporates optical flow with RGB images to capture both position and motion information. Therefore, our experimental conclusions should generally hold with other driving datasets and models.

The second external validity concern is the selection of baselines. To mitigate this, we compare UniAda with DeepManeuver [36], Perturbation Attack [43], DeepBillboard [27], and FGSM [30] that are either state-of-the-art (e.g., DeepManeuver, published in 2023) or representative (e.g., DeepBillboard, FGSM) in this field.

Construct Validity: Construct validity concerns whether the chosen evaluation metrics accurately capture the intended effect. In our study, our goal is to design an attack technique that generates image-agnostic perturbations capable of inducing errors

in the model under test. We mitigate the threat by examining the attack effectiveness on two metrics, mean error and success rate, which are widely used in the literature [101, 50, 28, 27]. The mean error can reflect the average attack strength, however, it might be biased by some outliers and cannot represent the performance of most input images. To counter this threat, we also measure the success rate, which manifests the universality of the attack (i.e., whether the generated perturbation can affect most input images) by examining the percentage of images that are successfully attacked under different thresholds.

3.7 Chapter Summary

In this chapter, we examine the reliability and security problems raised by adversarial attacks on three E2E ADSs. Our proposed method, UniAda, conducts novel and comprehensive testing by addressing three main limitations from existing literature: limited vehicle controls testing, constrained testing scenarios, and image-specific or unnatural perturbations. We alleviate them accordingly by (1) performing testing on steering and acceleration controls simultaneously through the multi-objective attack with Adaptive Weighting Scheme, (2) examining both simulated and real-world driving scenarios in urban traffic, and (3) generating image-agnostic (i.e., universal) and human-eye-imperceptible perturbation through joint optimization. The effectiveness of UniAda is compared with four baselines in mean error and success rate metrics. Compared with its closest competitor, DeepManeuver, UniAda achieves an improvement of 6.6°, 8.6°, and 2.9° in steering error on CILRS, CILR, and MT ADSs, respectively.

Chapter 4

Enhancing the Transferability of Adversarial Attacks for End-to-End Autonomous Driving Systems

Adversarial attacks play an important role in testing and enhancing the reliability of deep learning (DL) systems. Most existing attacks for DL-based autonomous driving systems (ADSs) demonstrate strong performance under the white-box setting but struggle with black-box transferability, while black-box attacks are more practical in real-world scenarios as they operate without full model access. Numerous transferability-enhancement techniques have been proposed in other fields (e.g., image classification), however, they remain unexplored for end-to-end (E2E) ADSs.

Our study fills the gap by conducting the first comprehensive empirical analysis of nine transferability-enhancement methods on E2E ADSs, covering two types: three input transformation enhancements and six attack objective enhancements. We evaluate their effectiveness on two datasets with four steering models. Our findings reveal that, out of nine enhancements, Resizing+Translation delivers the best black-box transferability, producing up to 9.39° increase in MAE. Pred+Attn serves as the best objective enhancement, producing a maximum of 5.55° (white-box) and 6.21° (black-box) increase in MAE. Through attention heatmap visualizations, we discover that different models focus on similar regions when predicting, thereby enhancing the transferability of attention-based attacks.

In conclusion, our study provides valuable results and insights into the transferability-enhancement techniques for E2E ADSs, which also serve as a robust benchmark for further advancements in the autonomous driving field. This work was

published in the 31st Asia-Pacific Software Engineering Conference, IEEE (APSEC 2024) [145].

The subsequent sections of this chapter are organized as follows: Section 4.1 introduces notations and foundations used in this chapter. The design of the empirical study is described in Section 4.2, including experimental procedure, research questions studied in this work, and experimental setup. Section 4.3 describes the results and analysis of the experiments. Section 4.4 presents the threats to the validity. Finally, Section 4.5 summarizes the work of this chapter.

4.1 Preliminaries

In this section, we define the notations used throughout this chapter and formalize the goals and capabilities of attackers to the targeted ADSs. Then, we provide a formulated definition of (1) adversarial attacks on the steering regression task and (2) Transferability-enhancement techniques employed in this paper (Input Transformation Enhancement and Attack Objective Enhancement).

4.1.1 Notations and Threat Model

• x_{ori}: the original input image.
• x_{adv}: the generated legitimate adversarial image, such that the pixel values must be within the range [0,1]. We denote as $x_{adv} = Process(x_{ori} + \tau)$.
• x: to represent the general image input.
• m: the white-box model that we craft the adversarial perturbation on. We denote $m(x) \in R$ as the predicted steering angle value with the input x.
• τ: the adversarial perturbation pattern.
• ϵ: the maximum perturbation.

Threat Model: We perform *iterative white-box* gradient-based attacks with specified attack objectives to generate an adversarial sample and use the transferability to conduct *black-box attacks* on other steering models for which we have limited access. Steering models take RGB images as input and output real continuous steering values. Specifically, we intentionally craft a malicious adversarial sample x_{adv} to fool the steering model m into making wrong steering decisions. Our attack is *image-specific* such that we generate a unique τ for each RGB image. The adversarial image x_{adv} is visually similar to the x_{ori} and conforms to the training distribution. Moreover, our attacks are *non-targeted*, in which we do not guide m towards a specified

prediction direction. The crafted x_{adv} can induce m to predict a steering value that is either larger or smaller than the original prediction $m(x_{ori})$.

4.1.2 Adversarial Attacks

Adversarial attacks in DNNs aim to generate a subtle, human-imperceptible perturbation τ that induces the model to make false predictions. In the steering regression task, the goal of adversarial attacks is maximizing the model prediction difference between the original and the adversarial input [50], expressed as $\text{argmax}_{\tau}\ |m(x_{adv}) - m(x_{ori})|$ subject to the imperceptibility constraint $\|\tau\|_p \leq \epsilon$. $\|\cdot\|_p$ could be L_1, L_2 or L_∞ norm (we use L_∞) [146].

A common objective to perform white-box gradient-based attacks is prediction-difference loss. We take the absolute value to give equal importance to different error scales:

$$L_{pred}(x; m) = -|m(x_{adv}) - m(x_{ori})| \tag{4.1}$$

By minimizing the loss, we are maximizing the difference between the prediction on the original and adversarial sample.

4.1.3 Transferability-Enhancement Techniques

Input Transformation Enhancement: Input Transformation is one of the most effective approaches to improve attack transferability. Crafting with diverse input patterns, the generated τ is expected to be less sensitive to the discriminative region of the white-box model being attacked [47, 51].

In this paper, we explore two input transformations that are commonly used in other fields [51, 47, 147, 148]: Random Resizing and Translation. Each transformation is applied to x during the perturbation generation process with the **prediction-difference loss** L_{pred} (Eq. 4.1) as the attack objective.

(1) Random Resizing [47] first resizes the input image to a random size and then randomly pads zeros around it (i.e., pads back to the original size). The transformation T is applied with probability $p \in [0, 1]$ at each attack iteration. Specifically, we use the transformed image $T(x)$ with probability p and the non-transformed image x with probability $1 - p$. In our experiments, we set $p = 0.5$, following the default settings in [47]. The input is initially randomly resized to $rnd \times rnd \times 3$ with $rnd \in [90, 100)$, and then it is padded back to the size $100 \times 100 \times 3$ with zeros.

(2) Translation operation [51] $T_{ij}(\cdot)$ shifts the image x by i and j pixels along the two-dimensions, respectively. In other words, each pixel (a, b) of the translated image is defined as $T_{ij}(x)_{a,b} = x_{a-i,b-j}$. We set $i, j \in \{-k, ..., 0, ..., k\}$ with $k \leq 10$ for small translations to hold the assumption for fast gradient calculation with kernel matrix. Following [51], we use the default Gaussian kernel with a kernel length of 15 and a radius of 3 for our experiments.

Attack Objective Enhancement: Another line of research attempts to enhance attack transferability by investigating model-agnostic features and disrupting them using novel loss functions during the perturbation generation process. We consider two well-known objective improvements: Feature-disruptive Loss and Attention-distractive Loss [110, 52]. In our experiments, we generate τ using images **without transformations** and employing different enhancement losses as attack objectives.

(1) Feature-disruptive Loss: Previous studies [55, 56] have revealed that shallow-layer features learned by different models are similar. Based on this observation, literature in both image classification [110] and perception [53, 54] proposed to craft the adversarial sample by maximizing the discrepancy in the intermediate feature between the clean sample and the perturbed sample. Specifically, the goal is achieved by amplifying the less significant channel-wise features while weakening the significant channel-wise features.

$$\begin{aligned} L_{feat}(x; m) = log(\sum_{i,j} \sum_{p \in S} |m_p^l(x) \odot \nabla m_p^l(x)|_{ij}) \\ - log(\sum_{i,j} \sum_{q \in \hat{S}} |m_q^l(x) \odot \nabla m_q^l(x)|_{ij}) \end{aligned} \tag{4.2}$$

where $\odot$ is the element-wise product, S and $\hat{S}$ represent the channel index set of significant and less significant channel-wise features, respectively (partitioned by channel-wise mean-based strategy described in [110]). $|\cdot|$ is the absolute value function. $m_p^l(x)$ refers to the feature map of p-th channel at the l-th layer of the model. $\nabla m_p^l(x)$ represents the gradient of the predicted steering angle with respect to the input image x at the p-th channel of the l-th layer of the model m. We sum over all (i, j) elements in the feature map and all channels in the index set, and then take the logarithm of it and compute the difference as the loss function. By minimizing the loss function, the generated perturbation causes important channel-wise features to become less important, while less significant features become more important. As suggested in [110], we select the first convolutional layer of each model for l.

(2) Attention-distractive Loss: Most steering models, including those we studied, employ CNN-based architectures to learn and extract local features from input images. Understanding the visual attention of these models is crucial to comprehend which areas they prioritize during predictions [149]. Previous studies have introduced efficient attention-guided adversarial samples in tasks like image classification and object detection. They also demonstrated the transferability effectiveness of the generated adversarial samples [111, 52, 150].

Following common practices [150, 151], we adopt Grad-CAM [149], originally designed for classification problems, to interpret predictions made by steering models. To locate the attention area, we first obtain the activation value A^k (i.e., feature map) of a specific convolution layer (usually the last convolution layer) through forward propagating the input image x. Then, we back-propagate the model prediction signal L_{cam} to A_k and obtain positive contribution regions.

Adapting Grad-CAM for regression tasks involves using the prediction signal L_{cam} instead of softmax values [1]. Specifically, we divide and adjust steering angles y into three ranges by δ. δ is a self-defined small positive threshold, where we select $\delta = 5^\circ$, which is relatively small according to the distribution of steering angle data. In this way, by taking the derivative of L_{cam}, the positive gradients always contribute to the model prediction, regardless of whether the current steering prediction is positive, negative, or nearly zero.

The resulting gradient values $\partial L_{cam}/\partial A^k_{ij}$ are then aggregated via global average pooling to derive the importance weight a_k of the feature map A^k for each channel k. The attention map $h(x, y)$ is then computed as the weighted sum of A^k:

$$L_{cam} = \begin{cases} y, & \text{if } y > \delta \\ -y, & \text{if } y < -\delta \\ \frac{1}{y} \times \text{sign}(y), & \text{otherwise} \end{cases} \quad \Rightarrow \quad \begin{array}{l} a_k = \frac{1}{Z} \sum_i \sum_j \partial L_{cam}/\partial A^k_{ij}, \\ h(x,y) = ReLU(\sum_k a_k A^k) \end{array} \tag{4.3}$$

We choose the feature map A^k of the last convolutional layer for each model, as suggested in [149]. We employ attention-distractive loss L_{attn} to distract the focus from the original attention map:

$$L_{attn}(x; m) = -\left\| \frac{h(x,y)}{max(h(x,y))} - \frac{h(x_{ori}, y_{ori})}{max(h(x_{ori}, y_{ori}))} \right\|_1 \tag{4.4}$$

[1] we follow https://jacobgil.github.io/deeplearning/vehicle-steering-angle-visualizations

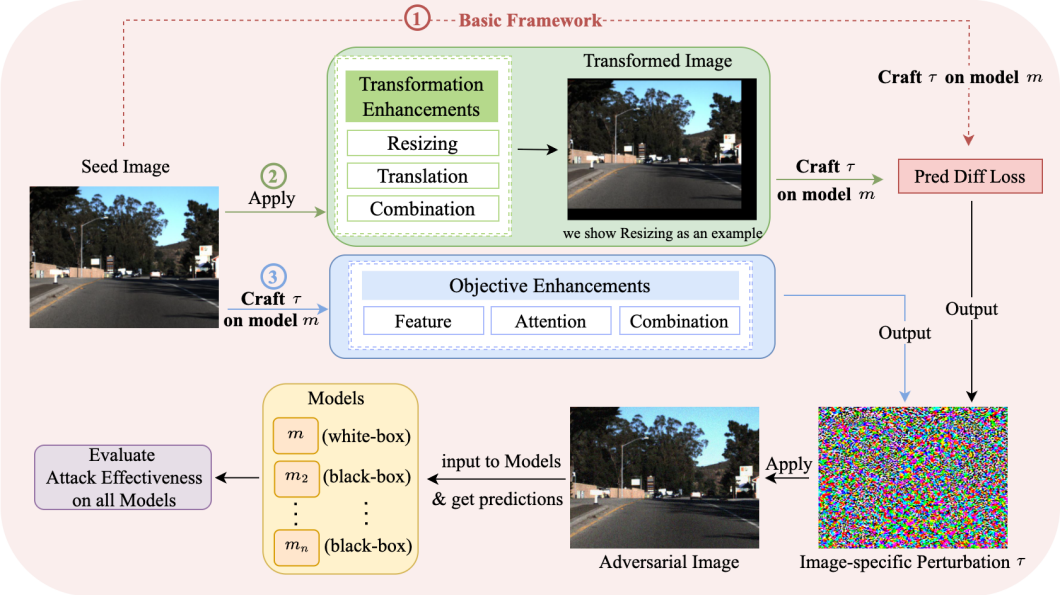


Fig. 4.1 The framework of adversarial attacks on steering models. Workflow 1: the Baseline Framework. Workflow 2: Input Transformation Enhancement. Workflow 3: Attack Objective Enhancement. m denotes the white-box model we craft the perturbation for. $m_2, ..., m_n$ denotes the black-box models being tested for transferability evaluation.

where $\|\cdot\|_1$ stands for the L_1 norm. Self-normalization is imposed to eliminate the influence of different magnitudes. By minimizing L_{attn}, we maximize the difference between the original attention map $h(x_{ori}, y_{ori})$ and the adversarial attention map $h(x, y)$.

4.2 Study Design

4.2.1 Experimental Procedure

Figure 4.1 illustrates the framework utilized in our study, containing three workflows. Each workflow is distinguished by color: the baseline framework (Workflow 1, red), the input transformation enhancement framework (Workflow 2, green), and the attack objective enhancement framework (Workflow 3, blue).

In the baseline framework, we generate adversarial perturbations without employing any transferability-enhancement techniques. We use the non-transformed image to iteratively search for the perturbation τ by attacking the white-box model m with prediction-difference loss minimization (Eq. 4.1). After that, we apply the

optimized τ to the seed image to obtain the adversarial image. Then, we evaluate the white-box and black-box attack effectiveness by computing the corresponding metric. The results of the baseline framework are served as the **baseline** in this paper.

When applying the input transformation enhancements (workflow 2, green), we use the transformed image to iteratively search for the perturbation by attacking the white-box model m with prediction-difference loss minimization. The optimized τ is then evaluated for attack effectiveness.

For attack objective enhancements (workflow 3, blue), we use the non-transformed image to iteratively search for the perturbation with enhanced attack objectives (i.e., feature-disruptive loss (Eq. 4.2), attention-distractive loss (Eq. 4.4), and combination losses). After the searching process, we obtain the perturbation τ and evaluate its white-box and black-box attack effectiveness.

Algorithm 3 Adversarial Perturbation Generation Process.

Require: x_{ori}: the seed image
Require: m: the white-box steering model
Require: lr: the learning rate
Require: ϵ: the maximum perturbation
Require: $iters$: the number of iterations
Require: the loss functions - $\{L_{pred}, L_{feat}, L_{attn}\}$
Require: T: input transformation - {Resizing, Translation, Combination}

$x = copy(x_{ori})$
$\tau = zeros(x.shape)$
for i in iters **do**
    if Workflow 1: Baseline Framework **then**
        $l = L_{pred}(x; m)$
    else if Workflow 2: Input Transformation Enhancement **then**
        $l = L_{pred}(T(x); m)$
    else if Workflow 3: Attack Objective Enhancement **then**
        $l = \alpha \times L_{pred}(x; m) + \beta \times L_{feat}(x; m) + \gamma \times L_{attn}(x; m)$
    end if
    $G = \text{sign}(\partial l / \partial x)$
    Update $\tau \leftarrow Clip_{\epsilon}(\tau - lr \times G)$
    Update $x \leftarrow Process(x_{ori} + \tau)$
end for
return τ

Algorithm 3 elaborates the details of the image-specific adversarial perturbation generation process. **Input:** The algorithm requires seven key inputs outlined in the **Require** lines. Specifically, x_{ori} denotes a single input seed image; m is the white-box

steering model that we know its parameters and architecture to perform the white-box attack; $lr, \epsilon, iters$ are hyperparameters, their values are listed and explained in the Section 4.2.3; The loss functions used as the objective to search for the perturbation; T is the input transformation. **Initialization:** Line 1 creates an image copy that requires gradients. Line 2 initializes the perturbation τ as a vector of zeros, with the same shape as the input RGB image. **Perturbation Generation:** For each iteration, it computes the loss l with different workflows (Line 4-10). Note that for Workflow 3, we set the weights of the other two losses as 0 when testing the individual loss. Then, we compute the gradient G and take the sign operator on it (Line 11). Line 12 updates the perturbation via gradient descent, with a maximum perturbation size of ϵ (L_∞ norm) to maintain imperceptibility. Afterward, the image x undergoes updates, subject to processing for legitimacy, for example, the pixel values should be within the range [0,255] (Line 13). **Output:** Upon reaching the maximum number of iterations, the algorithm yields an image-specific human-eye-imperceptible perturbation τ for the seed input x_{ori} (Line 15).

4.2.2 Research Questions

RQ1: *Which input transformation performs the best?*
We compare three input transformations (*Resizing*, *Translation*, *Resizing+Translation*) with *None* (i.e., adopt no input transformation, the baseline). We analyze their performance in both black-box (i.e., transferability) and white-box attacks under two public datasets.
RQ2: *Which attack objective yields the best performance. How does it compare to input transformations?*
We study seven loss functions, including three individual losses (the baseline L_{pred} in Eq. 4.1, L_{feat} in Eq. 4.2, and L_{attn} in Eq. 4.4) and four combinations ($L_{feat} + L_{attn}$, $L_{pred}+L_{feat}$, $L_{pred}+L_{attn}$, and $L_{pred}+L_{feat}+L_{attn}$). We analyze their performance in both black-box and white-box attacks for both datasets.
RQ3: *What is the runtime of studied techniques in generating a single adversarial perturbation?*
We examine the average runtime and its standard deviation of all studied techniques for both datasets for all models.

4.2.3 Experimental Setup

Datasets: Our experimentation uses two public datasets that are widely used in autonomous driving research [27, 28]: (1) the Udacity self-driving car challenge dataset [26], which contains 101,396 real-world driving images for training and 5614 images for testing, which is captured by a dashboard-mounted camera of a human-driven car. We use the Udacity test set for experimentation; (2) the Dave testing dataset [42], which contains 45,568 real-world driving images. The images are recorded from a dashcam with labeled steering angles, captured around Rancho Palos Verdes and San Pedro, California.

Models: We perform adversarial attacks separately on four normally trained steering models, which have been widely used in autonomous driving testing [27, 4, 36]. Specifically, we employ three CNN models based on the Dave2 architecture from NVIDIA, denoted as Dave2V1 [152], Dave2V2 [153], Dave2V3 [154], and one model (Epoch) from the Udacity challenge [155]. Specifically, Dave2V1 is the original model proposed by NVIDIA. Dave2V2 is a variation of Dave2V1, which initializes weights using a truncated normal continuous random variable. Dave2V3 is another variant that adds pooling layers between each convolutional layer. Epoch is a CNN-based steering model that ranked sixth in the final leaderboard of Udacity Self-driving Car Challenge 2 [2].

We employed the PyTorch re-implementation code provided by Von *et al.* [3]. Following the training process provided by the official Udacity GitHub page [4], we trained them on the Udacity train set with Mean Square Error (MSE) as the loss function, defined as $\frac{1}{N}\sum_{i=1}^{N}(m(x)-y)^2$. y denotes the ground truth steering label for the input image x and N denotes the number of images. We evaluated the model performance on the Udacity test set, where Dave2V1, Dave2V2, Dave2V3, and Epoch produce an MAE of 0.022, 0.028, 0.023, and 0.015, respectively.

Evaluation Metrics: To assess the attack effectiveness of the generated examples, we adhere to established practices and employ a widely used metric for the offline evaluation of steering models: Mean Absolute Error (MAE) [27, 36, 4, 156, 157]. Note that since we perform non-targeted attacks (see Section 4.1.1), we use the sign-invariant operator (absolute value) in the metric to avoid incorrect punishment for negative errors (situations where the attacker induces the model to predict a smaller steering than the original one). Let N denote the number of images:

[2] https://github.com/udacity/self-driving-car/tree/master/challenges/challenge-2

[3] https://github.com/MissMeriel/DeepManeuver/

[4] https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|m(x^i_{adv}) - m(x^i_{ori})| \tag{4.5}$$

A larger MAE indicates better attack effectiveness. To provide a more straightforward presentation of the attack effectiveness in the experiment results, we convert the steering values in MAE from radians to degrees (°) by $\times\frac{180}{\pi}$.

Hyperparameters: The max perturbation ϵ of each pixel in the image is set to 5 for all experiments. Following the optimal setting in Co *et al.* [158], the learning rate lr is set to 0.8, and the number of iterations is set to 3. For the weights in the combination losses, we set $\alpha = 1$ as the reference magnitude for L_{pred}. For β and γ, we conduct a grid search on a small subset of the dataset from candidate values $\beta \in \{0.1, 0.3, 0.5, 0.7, 1\}$ and $\gamma \in \{1e-6, 1e-7, 1e-8, 1e-9\}$, and choose the best combination for experimentation. Small values for γ are selected to ensure three losses can have a similar variance for different inputs [52]. Specifically, we select $\beta = 0.7$ and $\gamma = 1e-8$. For the hyperparameters related to the individual enhancements, we adhere to the optimal settings described in the original paper and list the values as introduced in Section 4.1.

4.3 Results and Analysis

In this section, we present the results of our experiments to answer RQ1, RQ2, and RQ3. Moreover, we provide insights into the superior performance of the studied transferability-enhancement techniques.

4.3.1 RQ1. Input Transformation Enhancement

With L_{pred} as the attack objective, we perform adversarial attacks under three different input transformation enhancements (*Resizing*, *Translation*, and *Resizing+Translation*) on two datasets. We also show the results of no transformation *None* (Workflow 1, Baseline Framework) as the baseline for comparison. We craft the adversarial perturbation on each white-box model and test the perturbation on all models. Each cell in Table 4.1 indicates the MAE result in degrees. The first column shows the white-box models used to generate the adversarial perturbation, and the first row denotes the model that we test the perturbation on. Larger MAEs indicate better performance.

Overall, we observe a consistent performance trend for both datasets from Table 4.1. For **black-box** attacks, under both datasets, *Resizing+Translation* outperforms

Table 4.1 MAE results (in degrees) of Input Transformation Enhancement (with L_{pred} as the attack objective). The superscript * indicates the white-box attack, and the others are black-box attacks (i.e., transferability). We highlight the results of the best technique in bold.

	Input Transformation	Udacity Dave2V1	Dave2V2	Dave2V3	Epoch
Dave2V1	None (Baseline Framework)	**18.40***	4.11	1.95	1.75
	Resizing	16.37*	4.01	1.95	1.83
	Translation	16.39*	5.21	2.29	2.24
	Resizing+Translation	17.50*	**5.49**	**2.34**	**2.31**
Dave2V2	None (Baseline Framework)	3.64	**14.26***	1.93	1.69
	Resizing	3.84	12.43*	2.06	1.83
	Translation	4.55	10.89*	2.98	2.64
	Resizing+Translation	**6.53**	10.46*	**3.03**	**2.73**
Dave2V3	None (Baseline Framework)	0.34	0.36	**52.21***	0.58
	Resizing	0.37	0.36	31.11*	0.63
	Translation	2.36	2.32	19.61*	3.02
	Resizing+Translation	**4.05**	**2.43**	15.91*	**3.11**
Epoch	None (Baseline Framework)	1.71	2.98	4.87	**16.21***
	Resizing	3.38	2.98	4.90	14.32*
	Translation	4.56	4.18	5.69	13.06*
	Resizing+Translation	**7.47**	**4.54**	**6.06**	12.68*
	Input Transformation	Dave Dave2V1	Dave2V2	Dave2V3	Epoch
Dave2V1	None (Baseline Framework)	**26.48***	4.57	2.65	3.35
	Resizing	23.26*	4.57	2.71	3.42
	Translation	24.12*	5.49	2.86	3.48
	Resizing+Translation	22.42*	**5.66**	**2.93**	**3.72**
Dave2V2	None (Baseline Framework)	5.23	**17.37***	2.45	2.33
	Resizing	5.08	14.78*	2.46	2.34
	Translation	6.53	13.92*	3.88	3.45
	Resizing+Translation	**8.86**	13.46*	**7.41**	**5.03**
Dave2V3	None (Baseline Framework)	0.55	0.42	**69.91***	1.54
	Resizing	0.54	0.43	49.05*	1.67
	Translation	2.80	2.72	31.51*	5.86
	Resizing+Translation	**3.40**	**4.05**	31.72*	**7.75**
Epoch	None (Baseline Framework)	3.96	2.75	5.79	**35.98***
	Resizing	3.86	2.80	5.92	31.70*
	Translation	5.46	3.91	7.50	26.33*
	Resizing+Translation	**7.77**	**10.26**	**15.18**	31.64*

all other methods by a decent margin in all cases. Compared to the baseline *None*, it improves transferability by producing a 0.39° to 5.76° (0.28° to 9.39°) increase in MAE for Udacity (Dave). *Translation* achieves the second best results, which outperforms *Resizing* and *None* in all cases, with a 0.34° to 2.85° (0.13° to 4.32°) improvement in MAE compared to the baseline *None* for Udacity (Dave). In 66.6% cases, *Resizing* outperforms the baseline, with the improvement in MAE up to 1.67° (0.13°) for Udacity (Dave). Diving into individual cases, taking the adversarial perturbation crafted on Epoch and test on Dave2V1 (Epoch-Dave2V1) as an example, the *Resizing+Translation* results in an MAE of 7.47° under Udacity, which is 5.76°, 4.09°, and 2.91° larger than the MAEs caused by *None*, *Resizing*, and *Translation*, respectively. For the Dave dataset, taking Epoch-Dave2V3 as an example, the *Resizing+Translation* causes an MAE of 15.18°, where *Translation*, *Resizing*, and *None* only result in MAEs of 7.50°, 5.92°, and 5.79°, respectively.

In short, we suggest applying *Resizing+Translation* if the attacker's goal is to attack black-box models.

Finding 1: For black-box attacks, across two datasets, *Resizing+Translation* achieves the best performance. Compared to the baseline, the improvement in MAE ranges from 0.28° to 9.39°. *Translation* produces the second-best results, with the improvement in MAE ranging from 0.13° to 4.32°. Only 66.6% of the time *Resizing* outperforms the baseline.

For **white-box** attacks, the baseline *None* performs the best in all cases across both datasets compared to the input transformation enhancements. In 6 out of 8 cases, *Resizing* outperforms *Translation*. *Resizing+Translation* shows the poorest attack effectiveness in 5 out of 8 cases. Specifically, the MAE of the baseline can reach up to 52.21° (69.91°) for Udacity (Dave). *Resizing* demonstrates a decrease in MAE compared to the baseline, ranging from 1.83° to 21.1° (2.59° to 20.86°) for Udacity (Dave). Similarly, the decrease in MAE under *Translation* ranges from 2.01° to 32.6° (2.36° to 38.4°) for Udacity (Dave). We observe the same phenomenon in *Resizing+Translation*, with 0.9° to 36.3° (3.91° to 38.19°) decrease in MAE under Udacity (Dave). With input transformations, the adversarial perturbation is crafted based on an ensemble of transformed and non-transformed images, which may tend to underfit the white-box model parameters and result in inferior white-box performance. Therefore, we suggest using no transformation for white-box attacks.

Finding 2: All transformation methods degrade the white-box attack performance compared to the baseline *None*, with a decrease in MAE ranging from 0.9° to 38.4°.

4.3.2 RQ2. Attack Objective Enhancement and its Comparison with Input Transformation

We use seven losses separately to craft adversarial perturbations on each of the four white-box models and test the generated perturbations on all models. The MAE results (in degrees) are reported in Table 4.2 and Table 4.3. The first column shows the white-box models on which we craft the perturbation. The first row denotes the model that we test the perturbation on. Specifically, the diagonal blocks (numbers with asterisks) under each dataset indicate the white-box attack performance, and others indicate the black-box attack performance. Larger MAEs indicate better performance. For simplicity, we use *Pred*, *Feat*, *Attn* to denote L_{pred} (Workflow 1), L_{feat}, and L_{attn}, respectively.

For **white-box** attacks, *Pred+Attn* demonstrates the best effectiveness in 6 out of 8 cases, reaching up to 5.55° improvements over the baseline in MAE. We observe that, in 7 out of 8 cases, generating the perturbation with *Feat* or its combination loss (*Feat+Attn* or *Pred+Feat*) results in the worst white-box performance. Especially, *Feat+Attn* leads to a surprisingly poor white-box performance in many cases, such as Dave2V3-Dave2V3 under Udacity.

For transferability on **black-box** models, compared to the baseline, *Pred+Attn* reaches the best performance in 18 out of 24 cases, achieving 0.08° to 6.21° improvements. *Feat*, *Feat+Attn*, and *Pred+Feat* do not produce outstanding performance in most cases for both datasets. Specifically, in 13 out of 24 cases, *Feat+Attn* outputs the worst performance on black-box attacks, producing a maximum decrease of 4.51°.

Finding 3: For both white-box and black-box attacks, we suggest using *Pred+Attn* as the attack objective, producing a maximum of 5.55° (white-box) and 6.21° (black-box) improvements in MAE compared to the baseline. We do not suggest using *Feat+Attn* for either attack type.

We then compare the best input transformation *Resizing+Translation* for transferability enhancement with the best objective *Pred+Attn*. For black-box transferability, the former outperforms the latter in 21 out of 24 cases, with an

Table 4.2 MAE results (in degrees) of Attack Objective Enhancement (with NO input transformation). The superscript * indicates the white-box attack, and the others are black-box attacks (i.e., transferability). We highlight the results of the best technique in bold.

	Attack Objective	Udacity			
		Dave2V1	Dave2V2	Dave2V3	Epoch
Dave2V1	Pred (Baseline Framework)	18.40*	4.11	1.95	1.75
	Feat	7.10*	2.65	1.54	1.35
	Attn	18.82*	4.29	1.78	1.81
	Feat+Attn	2.24*	1.28	1.64	1.24
	Pred+Feat	9.62*	3.34	1.63	1.16
	Pred+Attn	**23.95***	**4.71**	**2.25**	**2.10**
	Pred+Feat+Attn	17.80*	4.34	2.10	1.92
Dave2V2	Pred (Baseline Framework)	3.64	14.26*	1.93	1.69
	Feat	2.39	5.06*	1.77	1.21
	Attn	**4.08**	13.97*	1.84	1.74
	Feat+Attn	0.88	2.61*	1.20	0.80
	Pred+Feat	1.24	2.40*	1.41	0.84
	Pred+Attn	3.85	**15.31***	**2.15**	**3.30**
	Pred+Feat+Attn	3.04	4.82*	1.13	1.14
Dave2V3	Pred (Baseline Framework)	0.34	0.36	**52.21***	0.58
	Feat	0.43	0.36	30.26*	0.61
	Attn	0.37	0.39	50.02*	0.57
	Feat+Attn	0.43	0.29	2.64*	0.63
	Pred+Feat	0.46	0.45	50.00*	0.74
	Pred+Attn	**0.75**	**0.46**	47.15*	0.66
	Pred+Feat+Attn	0.42	0.39	43.81*	**0.82**
Epoch	Pred (Baseline Framework)	1.71	2.98	4.39	16.21*
	Feat	1.78	2.19	3.02	8.28*
	Attn	1.98	2.86	4.07	16.10*
	Feat+Attn	0.43	0.80	0.91	1.67*
	Pred+Feat	4.91	3.15	4.20	12.75*
	Pred+Attn	**6.06**	**3.38**	**4.89**	**17.34***
	Pred+Feat+Attn	5.32	3.28	4.43	13.91*

Table 4.3 MAE results for the Dave dataset.

	Attack Objective	Dave			
		Dave2V1	Dave2V2	Dave2V3	Epoch
Dave2V1	Pred (Baseline Framework)	26.48*	4.57	2.65	3.35
	Feat	14.09*	3.83	2.46	2.18
	Attn	6.11*	4.15	**3.87**	3.23
	Feat+Attn	8.10*	3.01	2.56	2.83
	Pred+Feat	16.49*	5.27	2.93	2.88
	Pred+Attn	**27.94***	**6.37**	3.20	3.46
	Pred+Feat+Attn	27.72*	6.15	3.09	**4.94**
Dave2V2	Pred (Baseline Framework)	5.23	17.37*	2.45	2.33
	Feat	2.29	3.08*	2.22	2.42
	Attn	2.49	6.87*	3.06	2.90
	Feat+Attn	2.07	3.68*	1.31	0.92
	Pred+Feat	1.19	3.45*	2.71	2.18
	Pred+Attn	**7.48**	**22.68***	**5.19**	**3.93**
	Pred+Feat+Attn	3.87	11.06*	2.80	1.82
Dave2V3	Pred (Baseline Framework)	0.55	0.42	**69.91***	1.54
	Feat	0.47	0.40	36.08*	0.61
	Attn	0.29	0.32	22.97*	1.23
	Feat+Attn	0.39	0.43	3.82*	**2.08**
	Pred+Feat	0.61	**0.62**	55.66*	0.89
	Pred+Attn	**0.72**	0.50	63.12*	1.90
	Pred+Feat+Attn	0.59	0.48	53.19*	1.86
Epoch	Pred (Baseline Framework)	3.96	2.75	5.79	35.98*
	Feat	2.70	2.57	4.45	19.82*
	Attn	0.75	1.53	2.38	3.36*
	Feat+Attn	0.66	1.40	1.28	1.84*
	Pred+Feat	5.17	4.28	7.95	33.29*
	Pred+Attn	**6.91**	**7.17**	**12.00**	**39.83***
	Pred+Feat+Attn	6.00	5.45	7.62	38.95*

Table 4.4 The average runtime (in seconds) of perturbation generation and its standard deviation for Udacity.

Method	Basic	Resize	Trans	Resize+Trans	Feat
Dave2V1	0.01±0.002	0.01±0.002	0.05±0.010	0.05±0.008	0.04±0.001
Dave2V2	0.01±0.002	0.02±0.001	0.04±0.011	0.05±0.008	0.04±0.002
Dave2V3	0.01±0.001	0.01±0.001	0.03±0.009	0.03±0.007	0.03±0.002
Epoch	0.01±0.001	0.01±0.001	0.11±0.005	0.11±0.003	0.03±0.002
Method	Attn	Feat+Attn	Pred+Feat	Pred+Attn	Pred+Feat+Attn
Dave2V1	0.09±0.012	0.39±0.020	0.05±0.002	0.09±0.010	0.37±0.029
Dave2V2	0.09±0.010	0.38±0.020	0.04±0.002	0.09±0.011	0.38±0.027
Dave2V3	0.08±0.006	0.37±0.023	0.03±0.001	0.10±0.005	0.36±0.011
Epoch	0.14±0.016	0.41±0.017	0.03±0.002	0.14±0.012	0.42±0.017

average of 1.61° improvement in MAE. In white-box attacks, *Pred+Attn* consistently outperforms Resizing+Translation, achieving an improvement of 12.69° in MAE on average.

Finding 4: On average, *Resizing+Translation* outperforms *Pred+Attn* in black-box attacks, showing a 1.61° increase in MAE. Conversely, *Pred+Attn* outperforms *Resizing+Translation* in white-box attacks, demonstrating a 12.69° increase in MAE.

4.3.3 RQ3. Runtime Efficiency Analysis

In RQ1 and RQ2, we have validated that adversarial samples generated with both types of enhancements demonstrate superior performance over the baseline for black-box transferability. To implement techniques into practice, efficiency is also an important factor to consider. For example, to realize real-time attacks, the attacker needs to timely generate an effective perturbation when encountering the seed input. With this consideration, we compute the average runtime (in seconds) and the standard deviation (std) of crafting a single adversarial perturbation on each of the four models for the Udacity and Dave datasets for all studied techniques.

Table 4.4 and Table 4.5 demonstrate the average runtime for the Udacity and Dave datasets, respectively. For both datasets and all models, the baseline framework takes the shortest time to generate an adversarial perturbation for the input image, with only 0.01s on average and a tiny std. *Feat+Attn* and *Pred+Feat+Attn* take similar

Table 4.5 The average runtime (in seconds) of perturbation generation and its standard deviation for Dave.

Method	Basic	Resize	Trans	Resize+Trans	Feat
Dave2V1	0.01±0.001	0.02±0.001	0.05±0.008	0.05±0.011	0.04±0.003
Dave2V2	0.01±0.001	0.02±0.001	0.05±0.012	0.06±0.009	0.04±0.004
Dave2V3	0.01±0.001	0.01±0.001	0.06±0.050	0.04±0.007	0.03±0.003
Epoch	0.01±0.001	0.01±0.001	0.12±0.023	0.11±0.004	0.03±0.002
Method	Attn	Feat+Attn	Pred+Feat	Pred+Attn	Pred+Feat+Attn
Dave2V1	0.09±0.003	0.40±0.028	0.05±0.002	0.10±0.004	0.39±0.023
Dave2V2	0.08±0.005	0.39±0.023	0.04±0.002	0.09±0.009	0.40±0.030
Dave2V3	0.07±0.005	0.34±0.020	0.04±0.001	0.07±0.004	0.39±0.020
Epoch	0.14±0.015	0.42±0.021	0.04±0.001	0.15±0.005	0.43±0.026

and the longest time, taking approximately 0.4s to generate a single perturbation for both datasets. Their std is also higher than other techniques. Among the four models, Epoch takes the longest runtime in 60% of cases, which may be attributed to its deeper architecture.

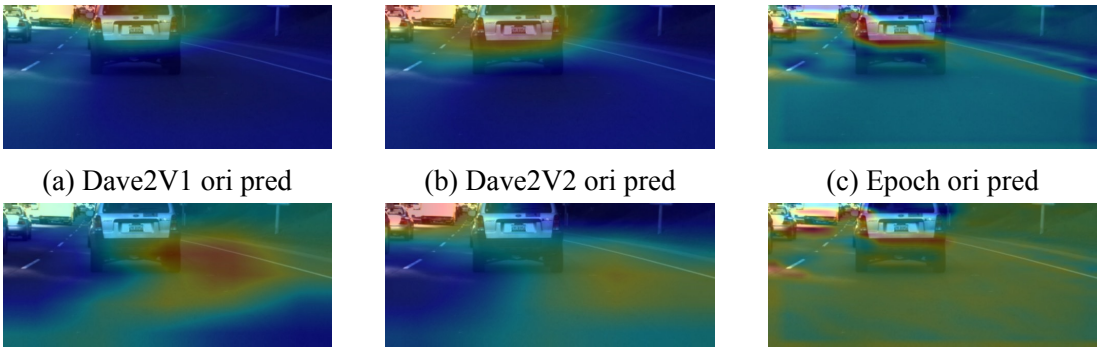
The best input transformation for black-box transferability *Resizing+Translation* takes only 0.06s on average to generate an adversarial sample, whereas the best objective *Pred+Attn* takes 0.11s, nearly double the time. Therefore, it is expected that *Resizing+Translation* can perform more timely attacks in real-world applications.

Finding 5: The best transformation *Resizing+Translation* (0.06s on average) for transferability demonstrates a nearly half runtime than the best objective *Pred+Attn* (0.11s).

4.3.4 Insights of Superior Performance

Based on the results from RQ1 and RQ2, we present insights into the superior performance produced by different transferability-enhancement techniques.

Input Transformation Enhancements. **Black-box Superiority:** With input transformation, an adversarial sample is crafted with an ensemble of images composed of the original one and its transformed versions. The generated adversarial sample will be less correlated to the gradient of the white-box model at the given input, which is expected to enhance the black-box transferability performance.



(a) Dave2V1 ori pred (b) Dave2V2 ori pred (c) Epoch ori pred

(d) MAE=44.8°,ϵ=5.0 (e) MAE=15.0°,ϵ=5.0 (f) MAE=9.60°,ϵ=5.0

Fig. 4.2 Attention Heatmap on Udacity sample image.

Attack Objective Enhancements. From the experiments, we identify *Pred+Attn* as the best attack objective for both white-box and black-box attacks. To explore the insights, we explore the relationship between the model's attention area and the model's prediction (i.e., can adversarial attacks change the model's attention?). We implement the baseline white-box attacks for simplicity and visualize attention heatmaps with and without attack in Figure 4.2.

White-box superiority of *Pred+Attn*: As shown in Figure 4.2a, Dave2V1 focuses on the pixel region around the front car when making steering predictions from the original image. However, its attention shifts to the lower-right region when predicting an adversarial image (Figure 4.2d). We can see that for each white-box attack, distracting the model's attention is complementary to distracting its prediction. Therefore, joint optimization of *Pred+Attn* achieves the best performance in white-box attacks 75% of the time.

Black-box superiority of *Pred+Attn*: In Figure 4.2a, 4.2b, 4.2c, all three models concentrate around the front car area for making steering predictions. The similarity in concentrated areas potentially encourages the black-box transferability of adversarial samples generated by distracting the white-box model's attention region. Our results in black-box attacks also prove this point.

4.4 Threats to Validity

Internal Validity: Threats to internal validity can be attributed to (1) the reproducibility of studied methods implemented on the E2E ADSs. To alleviate this, we carefully follow the publicly available code provided by the authors of original papers and open-source our implementation to facilitate checking; (2)

Effective manipulation of hyperparameters. To alleviate this, we follow the optimal hyperparameter configurations described in the original papers. For the weights in the combination objectives, we perform a comprehensive grid search with a sufficient number of candidate values (Section 4.2.3) on a small subset of the dataset to determine the optimal hyperparameter values.

External Validity: Threats to external validity to our experiment conclusions include (1) the selected datasets. We alleviate this by selecting two widely used datasets in autonomous driving research [4, 27, 28]. They cover various real-world driving scenarios, such as driving in urban towns, highways, and under different weather conditions.; (2) The selected E2E ADSs. We alleviate this by selecting four ADSs that reach competitive performance in the E2E steering angle prediction task [159, 153–155]. They are constructed with different architectures, ranging from 6 million (Dave2) to 33 million (Epoch) parameters. Therefore, our experimental conclusions are expected to generalize to other real-world driving datasets and models.

4.5 Chapter Summary

This chapter conducts an empirical study on transferability-enhancement techniques for E2E ADSs by examining three input transformations and six objective enhancements. Our findings indicate that Resizing+Translation performs the best in black-box attacks but shows degraded white-box performance. Using Pred+Attn as the attack objective improves both attack types. Additionally, our runtime efficiency assessment also demonstrates that Resizing+Translation takes only half the runtime of Pred+Attn, making it more practical for real-world black-box attacks. In future work, we plan to validate the effectiveness of the techniques in the physical world and design a framework that balances these trade-offs in terms of computational efficiency and effectiveness across different scenarios.

Chapter 5

Enhancing Valid Test Input Generation with Distribution Awareness for Deep Neural Networks

Comprehensive testing is important in improving the reliability of Deep Learning (DL)-based systems. Various Test Input Generators (TIGs) have been proposed to generate misbehavior-inducing test inputs. However, the lack of validity checking in TIGs often results in the generation of invalid inputs (i.e., out of the learned distribution), leading to unreliable testing. To save the effort of manually checking the validity and improve test efficiency, it is important to assess the effectiveness of automated validators and identify test selection metrics that capture data distribution shifts. In this study, we comprehensively assess four automated Input Validators (IVs) and six test selection metrics. Our findings show that the accuracy of IVs ranges from 49% to 77%. Distance-based IVs generally outperform reconstruction-based and density-based IVs for both classification and regression tasks. Additionally, we analyze the metrics' score achieved by valid and invalid inputs separately. The results reveal that invalid inputs consistently inflate uncertainty-based metrics.

Based on the findings, we enhance existing testing frameworks by incorporating distribution awareness through joint optimization. The results demonstrate our framework leads to a 2% to 10% increase in the number of valid inputs, which establishes our method as an effective technique for valid test input generation.

This work was published in the 48th IEEE International Conference on Computers, Software, and Applications [160].

The following sections of this chapter are structured as follows: Section 5.1 contains the first part of our work, the empirical study, including the empirical setup (the datasets and models studied), the TIGs, the IVs, and the test selection metrics studied and their association with input validity. Section 5.2 outlines our proposed approach based on the empirical findings and the result analysis. Section 5.3 presents threats to validity. Finally, Section 5.4 provides a summary of the chapter's work.

5.1 Empirical Study

Our pre-study analysis aims to identify effective IVs in validating test inputs generated by distinct TIGs. We further analyze the trustworthiness of existing test selection metrics in manifesting invalidity. Our findings in this section offer empirical evidence to enhance the existing testing framework, which is outlined in the following section. In summary, we have studied four automated IVs, three TIGs, and six test selection metrics, all of which are representative in the realm of DL testing and anomaly detection research.

5.1.1 Empirical Setup

The pipeline of our empirical setup is summarized in Figure 2.1. Initially, we randomly select 100 seed images from the dataset and pass them to each TIG to generate test inputs. The generated test inputs are then assessed by domain experts and IVs for validity identification. Binary validity labels (0 = invalid, 1 = valid) provided by human assessors are regarded as ground truth. With available ground truth, we assess the trustworthiness of automated validators and test selection metrics. All experiments were performed on machines equipped with an Intel Xeon Silver 4210 CPU, and an Nvidia GeForce RTX 3090-Ti GPU.

Datasets & Models. We evaluate two datasets, MNIST [89] and Udacity self-driving car challenge dataset [26], each with one DNN model that achieve competitive performance. **MNIST** contains 70000 grayscale images of handwritten digits from 0 to 9, each with dimension 28×28. There are 60000 training images and 10000 test samples. We use a classification model [8] with a 5-layer ConvNet with max-pooling and dropout layers, which achieves state-of-the-art performance (99.37% accuracy). **Udacity Driving Dataset** contains 101,396 training and 5614 testing RGB images captured by cameras mounted behind the windshield of a human-driving car. The steering angle applied by the human driver is also recorded along with each image

frame. We use the competitive DAVE-2 self-driving car model from Nvidia [152] with dropout layers, achieving 99.1% accuracy[1].

Human Evaluation. We analyze the validity of each generated test input from the human perspective. We invite 4 volunteers who are not coauthors of this paper to complete the questionnaire. All of the volunteers are PhD candidates and possess over four years of experience in the field of Artificial Intelligence and Software Engineering. Following the procedure from Riccio *et al.* [6], we designed the questionnaire with one question for each image generated by each testing technique.

We also include one attention check question (ACQ) [161] to filter out inattentive respondents and ensure the quality of the survey. For the test inputs on which two assessors disagree, we invite a third person to review the case and provide a majority vote. For the MNIST dataset, we ask the assessors to classify the image with eleven choices, including classes 0 to 9, "Not a handwritten digit", and "It is a digit but belongs to other classes, if yes, please specify" (e.g., number 11). The latter two choices indicate the input image is out of the valid input domain and should not be predicted by the model. For the Udacity dataset that captures real-world driving images, we ask whether the generated inputs are likely to be captured in real-world driving scenarios (e.g., can see the road clearly and can confidently steer the car under the given driving conditions).

5.1.2 Studied TIGs

Description. Existing TIGs [27, 7, 4, 11, 8] have been proven to efficiently generate a large number of misbehavior-inducing test inputs. However, they lack assessing the validity of the generated inputs, which cannot guarantee effective testing. This leads to the question:

Can TIGs Generate Invalid Inputs?

In this section, we analyze the quality of generated inputs from three different testing techniques with human evaluation: DeepXplore [4] with its three variants (differential testing), DLFuzz [7] (fuzz-based testing), and Sinvad [8] (search-based testing).

DeepXplore [4] is a white-box gradient-based test case generation technique with the training objective of maximizing the neuron coverage and differential behaviors of models. DeepXplore implements three different constraints on image modifications,

[1] we report 1-MSE (Mean Squared Error) as the accuracy.

namely lighting effects, occlusion by a small rectangle, and occlusion by multiple tiny black rectangles. We specifically follow its original implementation and generate inputs for all three constraints. We use the abbreviations DO for DeepXplore-occl, DL for DeepXplore-light, and DB for DeepXplore-blackout in the rest of the paper.

DLFuzz [7] (DF) is a white-box differential fuzzing testing framework to guide DL-based systems exposing incorrect behaviours. DF mutates the input sample by maximizing the neuron coverage and the prediction difference between the original and the mutated input. DF originally covered classification problems with the MNIST and ImageNet datasets. Specifically, it uses the prediction difference in softmax confidence as the objective to guide the input to cross the decision boundary of the original class. To extend it to the regression problem with the Udacity dataset, we use the actual prediction difference as the objective to guide the input generation.

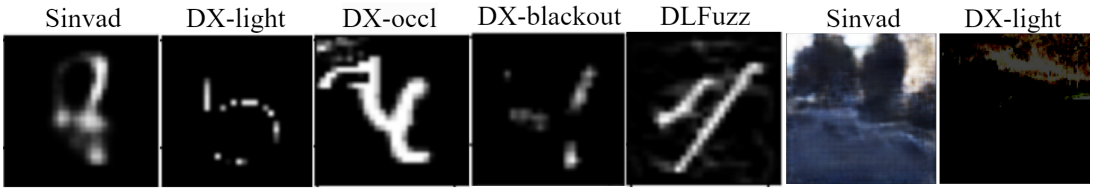
SINVAD [8] (SV) is a black-box test input generator. It searches over the latent space that resembles the true training distribution. Variational Autoencoder (VAE) is used to construct latent space, which is composed of an encoder and a decoder. Unlike DeepXplore, which navigates through and perturbs the original pixel space, SV applies perturbations on the latent space and then uses a decoder to reconstruct the image. We adapt this TIG to the Udacity dataset by integrating a Conv-based VAE with 10 hidden layers in the encoder and decoder, respectively, due to the greater complexity of driving images. To ensure a fair comparison, for each dataset, we use the *same* DNN model under test for all TIGs. For TIGs that are based on the TensorFlow-Keras framework, we convert the architecture of the PyTorch model under test to Keras and copy the model weights, and vice versa.

Observation. We examine the quality of generated test inputs with ground truth validity labels. The results are shown in the last column of Table 5.1.

For the MNIST dataset, all tested TIGs can produce invalid inputs. SV produces the highest number (48%) of invalid inputs. Techniques using pixel-level perturbations (DeepXplore with its variants and DF) tend to produce fewer invalid inputs. Perturbation in the feature space (e.g., SV) may lead to an amplified distortion in the pixel space through non-linear decoding, producing more distorted/invalid inputs. DB, the second least effective TIG, randomly adds small black holes on the seed image, which makes it easy to block the digit's white strokes and confuse the recognition, as shown in Figure 5.1(a). The rest three all generate $> 90\%$ valid inputs. DL can produce invalid inputs under excessive brightness or darkness. Figure 5.1(a) DX-light shows an example of diminished illumination. DF adds adversarial perturbation on the entire image, while DO adds an occlusion patch on the top left

Table 5.1 TIG Effectiveness: Input Validation results (%valid) with automated and human validators. For each IV, we highlight the best TIG in bold.

Dataset	TIG	DAIV	D-SVDD	D-KNN	PCNN	Human
MNIST	DO	2	32	81	25	96
	DL	86	95	60	86	**99**
	DB	**100**	90	60	**99**	80
	DF	26	91	**85**	0	92
	SV	**100**	**98**	81	14	52
Udacity	DO	95	**100**	97	76	**100**
	DL	89	80	74	**93**	75
	DB	79	**100**	**98**	90	**100**
	DF	63	**100**	85	0	**100**
	SV	**100**	**100**	**98**	25	0



(a) MNIST (b) Udacity

Fig. 5.1 Invalid test inputs for MNIST and Udacity datasets.

corner, potentially introducing disruptive white lines. For Udacity, all test inputs generated by DO, DB, and DF are valid, while none are for SV. We found that the test inputs generated by SV are excessively blurred, as depicted in Figure 5.1(b). This may also be attributed to the amplified distortions brought by feature-level perturbation. The first three introduce tiny modifications to the image pixels. Similar to MNIST, DL produces invalid inputs when there is excessive brightness or darkness.

Finding 1: All examined TIGs can generate invalid inputs. With human assessment, SV produces the highest number of invalid inputs for both datasets (48% for MNIST, 100% for Udacity). TIGs employing pixel-level perturbations yield a better performance.

5.1.3 Studied IVs

Description. We explore the effectiveness of density-based PixelCNN++ [68], reconstruction-based DAIV [64], distance-based DeepKNN [77] and DeepSVDD [79].

Following the established setting in the previous work [64, 6], we train all the IVs on the training set of each dataset, which is regarded as valid data by construction.

In this section, we provide a formulated description of each IV, followed by our experimental observations to the question:

Which IV is Better?

Specifically, with the ground truth labels provided by human assessment, for each IV, we compute True Positive (TP, valid for both automated and human validators), False Positive (FP, valid for automated IV but invalid for humans), True Negatives (TN, invalid for both automated and human validators), False Negatives (FN, invalid for automated IV but valid for humans), and Accuracy.

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

These evaluation metrics offer a comprehensive measurement of each IV's performance. For the formulated description, we use common notations t to denote the threshold, and x to denote the test input generated.

PixelCNN++ [68] is a generative model that explicitly models the statistical probability distribution on the training data. It uses a discretized logistic mixture likelihood on the pixels to speed up training. With each input image x, we can obtain its estimated probability density p with respect to the distribution modeled from the training data. Higher density indicates it is very likely that the input follows the training distribution (i.e., valid).

$$\log p = \sum_i \log p(x_i | x_{<i}) \Rightarrow x = \begin{cases} \text{valid}, & \text{if } \log p \geq t \\ \text{invalid}, & \text{if } \log p < t \end{cases} \tag{5.2}$$

i indicates all sub-pixels in x. To determine the threshold for normality, we follow common procedure [65, 77] to set the threshold at where 95% of the test data are classified as valid.

For MNIST, we apply 4 ResNet blocks consisting of 60 filters. For Udacity, we use a larger model with 5 ResNet blocks with 160 filters. The architecture for the decoding part is the same as the encoding part described above.

DAIV [64] is a VAE-based detector that trains a probabilistic encoder f_ϕ-decoder g_θ by minimizing the KL divergence of the posterior and prior distribution of the latent variable and maximizing the likelihood estimation of the input sample. The input sample with low reconstruction probability is classified as invalid. Specifically, the

encoder f_ϕ parameterizes a normal distribution with parameters μ_z, σ_z in the latent space with latent variable z, such that $z \sim N(\mu_z, \sigma_z)$. Then, we draw $z_l, l = 1, ...L$ samples from the normal distribution and send them to g_θ to output μ_{z_l}, σ_{z_l}. The reconstruction probability p for input x is computed as:

$$p = \frac{1}{L}\sum_{l=1}^{L} p_\theta(x|\mu_{z_l}, \sigma_{z_l}) \Rightarrow x = \begin{cases} \text{valid}, & \text{if } p \geq t \\ \text{invalid}, & \text{if } p < t \end{cases} \tag{5.3}$$

DAIV uses a self-defined invalid dataset to determine the threshold. The author calculates the reconstruction probability of the invalid dataset and picks the threshold value with the highest F-measure for identifying invalid inputs. Since DAIV originally covered MNIST and SVHN datasets only. We extend it to the Udacity dataset by applying a Conv-based VAE with 10 hidden layers in the encoder and decoder, respectively, due to the greater complexity of driving images. We create a synthetic invalid dataset for Udacity by distorting images with noise [162].

DeepKNN [77] explores the non-parametric nearest neighbor distance on feature space for anomaly detection. The non-parametric setting gives more flexibility by not imposing any distributional assumption of the underlying feature space. It computes the k-th nearest neighbor (KNN) distance between the embedding of the new test input and the embeddings of the training data. Then, a threshold-based criterion is used to detect whether the input sample is invalid. Specifically, let ϕ be a feature encoder, and

$$z = \frac{\phi(x)}{\|\phi(x)\|_2} \tag{5.4}$$

We determine the input validity by:

$$d_k = \|z - z^k_{train}\|_2 \Rightarrow x = \begin{cases} \text{valid}, & \text{if } d_k \leq t \\ \text{invalid}, & \text{if } d_k > t \end{cases} \tag{5.5}$$

where z^k_{train} is the normalized feature vector of the test input's k-th nearest neighbor in the training data. We follow the original paper and set t at which 95% of the valid data (we use the test set) to be correctly classified. Referring to the validation method used in Hendrycks *et al.* [162], we set $k = 50$ for MNIST and $k = 100$ for Udacity, selected from candidates $k = [1, 10, 20, 50, 100, 200, 500, 1000, 3000, 5000]$.

DeepSVDD [79] trains a DNN to encode the representations of the data into a hypersphere. The training objective is to distill common representations of valid data by minimizing the volume of the hypersphere. Specifically, DeepSVDD first pre-trains

the model via an autoencoder and then further trains the model initialized with the pre-trained weights. We follow the original implementation for the MNIST dataset and use a 10-layer Conv-based autoencoder for Udacity. We determine the input validity by computing the distance d_c between the test input x and the center c of the hypersphere:

$$d_c = \|\phi(x) - c\|_2 \Rightarrow x = \begin{cases} \text{valid}, & \text{if } d_c \leq t \\ \text{invalid}, & \text{if } d_c > t \end{cases} \tag{5.6}$$

c is initialized with the mean feature of the training set. We use the final radius of the trained model as the threshold value t. A new input is deemed invalid if its encoded feature by ϕ is outside the hypersphere.

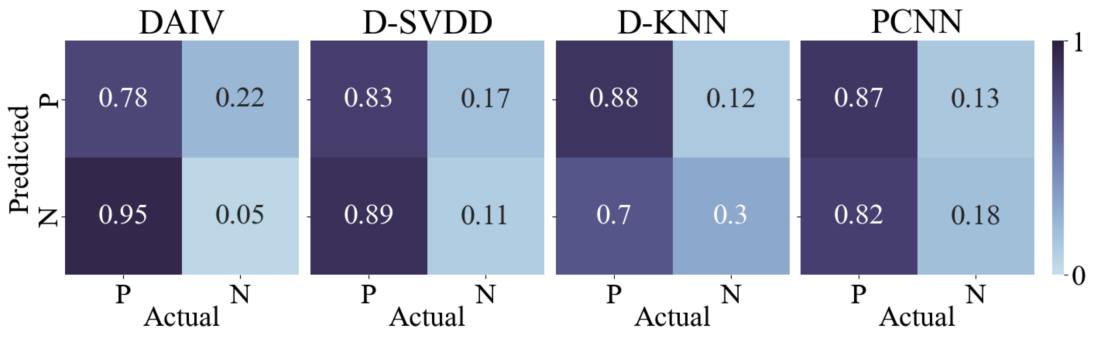
Observation. We demonstrate the validity result of each generated test suite with automated IVs and analyze the accuracy of each IV by comparing it with manual labels. **TIGs performance with IVs:** Table 5.1 shows the percentage of valid inputs identified by the four IVs. SV is identified as producing the highest number of valid misbehavior-inducing inputs over 60% cases, however, many of them are deemed invalid for humans. 50% of the time, DB produces the highest number of valid inputs.

Finding 2: Over 60% of the time IVs identify SV as the most effective TIG, which however contains many false positives compared to human assessments.

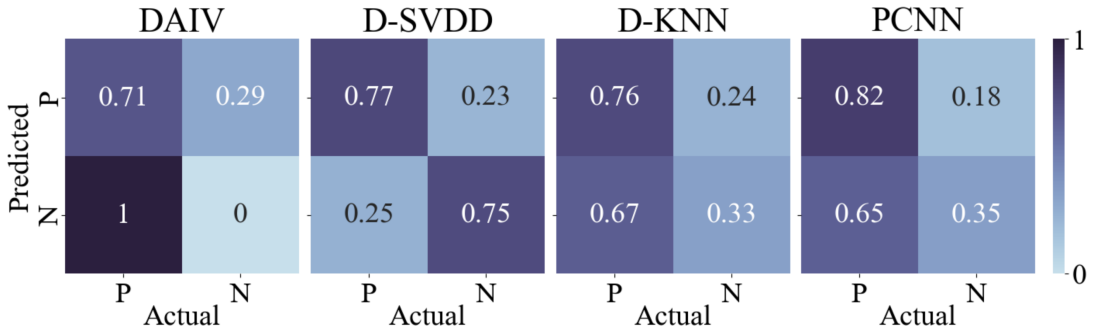
IV Effectiveness and recommendations: Table 5.2 provides the results of TP, FP, TN, FN, and Accuracy for measuring whether automated IVs reach an agreement with humans. Figure 5.2 provides an intuitive illustration of the average prediction summary of each IV, where a larger value in the diagonal indicates a better performance (i.e., higher TP and TN). For MNIST, on average of all generated test cases, D-KNN achieves the highest accuracy (72%). Similarly, distance-based D-SVDD takes the second spot. However, the average result is not uniform and the best IV varies under different TIGs. From Figure 5.2(a) MNIST, all IVs yield a high TP (upper left) but also a notably high FN (bottom left). A high FN indicates many valid test inputs are classified as invalid. For Udacity, the average accuracy indicates that feature-based IVs (D-SVDD and D-KNN) outperform DAIV and PCNN. Specifically, D-SVDD reach the highest accuracy under 4 out of 5 cases with an average of 77%. Intuitively, for image data with higher complexity, it is more difficult to directly model the distribution on pixel space (PCNN) or reconstruct the image (DAIV). Another interesting phenomenon is that all IVs except for PCNN perform poorly on Sinvad, with only 0% to 1% accuracy. Similarly to MNIST, Figure 5.2(b) Udacity also depicts

Table 5.2 IV Effectiveness: IV vs. human labeling. For each TIG, we highlight the best IV's Acc in bold. AV denotes the average. Abbreviations for IVs: DV for DAIV, SD for DeepSVDD, KN for DeepKNN, and PN for PixelCNN++.

IV	TIG	MNIST					Udacity				
		TP	FP	TN	FN	Acc	TP	FP	TN	FN	Acc
DV	DO	2	0	4	94	6	95	0	0	5	95
	DL	85	1	0	14	85	64	25	0	11	64
	DB	80	20	0	0	**80**	79	0	0	21	79
	DF	24	2	6	68	30	63	0	0	37	63
	SV	52	48	0	0	52	0	100	0	0	0
	AV	49	14	2	35	51	60	25	0	15	60
SD	DO	30	2	2	66	32	100	0	0	0	**100**
	DL	95	0	1	4	**96**	69	11	14	6	**83**
	DB	74	16	4	6	78	100	0	0	0	**100**
	DF	83	8	0	9	83	100	0	0	0	**100**
	SV	52	46	2	0	54	0	100	0	0	0
	AV	67	14	2	17	69	74	22	3	1	77
KN	DO	80	1	3	16	**83**	97	0	0	3	97
	DL	60	0	1	39	61	65	9	16	10	81
	DB	54	6	14	26	68	98	0	0	2	98
	DF	82	3	5	10	**87**	85	0	0	15	85
	SV	48	33	15	4	**63**	0	99	1	0	1
	AV	65	9	8	19	72	69	22	3	6	72
PN	DO	24	1	3	72	27	76	0	0	24	76
	DL	85	1	0	14	85	68	25	0	7	68
	DB	79	20	0	1	79	90	0	0	10	90
	DF	0	0	8	92	8	0	0	0	100	0
	SV	6	8	40	46	46	0	25	75	0	**75**
	AV	39	6	10	45	49	47	10	15	28	62



(a) MNIST



(b) Udacity

Fig. 5.2 The normalized confusion matrix depicting the average TP, FP, TN, FN values of all test suites for each dataset.

a high TP in all cases and a high FN in 3 out of 4 cases. D-SVDD stands out as an exception, demonstrating superior performance in accurately classifying both valid and invalid inputs.

Finding 3: The average accuracy of IVs ranges from 49% to 77%. Generally, we suggest using feature-based IVs for datasets with higher complexity. On average, D-KNN and D-SVDD achieve the best performance for MNIST and Udacity, respectively.

5.1.4 Studied Test Selection Metrics

Description. We examine six test selection metrics from three popular categories, including NC-based, SA-based, and uncertainty-based. The goal is to assess their association with input validity and answer the question:

Which Metric Can Better Capture Data Distribution Shift and Manifest Invalidity?

In this section, we analyze the average score of each metric for valid and invalid test inputs, respectively, as illustrated in Table **??**. For NC-based metrics, we discuss Neuron Coverage proposed by Pei *et al.* [4] and K-multisection Neuron Coverage proposed by Ma *et al.* [5]. For SA-based metrics, we selected Likelihood-based SA and Distance-based SA, both proposed by Kim *et al.* [15]. For uncertainty-based metrics, we examine DeepGini [13] for the MNIST classification model and MC-Dropout [131] for the Udacity regression problem.

Neuron Coverage [4] computes the ratio of neurons in a given DNN that are activated above a self-defined threshold value by a given set of test inputs T.

$$NC(T) = \frac{|\{n|\forall x \in T, a(n,x) > t\}|}{|N|} \tag{5.7}$$

$N = \{n_1, n_2, ...\}$ denotes all neurons in the DNN model under test. $a(n,x)$ is the neuron activation value produced by test input x on neuron $n \in N$. t is the self-defined neuron activation threshold. We set $t = 0.25$, which is commonly used in Deepxplore and DLFuzz.

K-multisection Neuron Coverage (KMNC) [5] divides coverage for each neuron into K sections. For a given test suite T, KMNC of a DNN measures the ratio of the number of sections covered by T for all neurons within the DNN under test.

$$KMNC(T,K) = \frac{\sum_{n \in N} |\{S_k^n | \exists x \in T : \phi(x,n) \in S_k^n\}|}{K \times |N|} \tag{5.8}$$

where $k \in \{1, ..., K\}$, S_k^n indicates the set of values in the k^{th} section. We use $K = 100$.

Likelihood-based Surprise Adequacy (LSA) [15] uses Kernel Density Estimation (KDE) to estimate the probability density of each activation value in the selected activation layer. Intuitively, it measures the surprise of input as the difference in the DNN's behavior (i.e., activation traces) between a new input and the training data. By using all training data as the reference, we can measure the surprise of the test input to the valid data distribution.

$$f(x) = \frac{1}{|\mathbf{X}|} \sum_{x_i \in \mathbf{X}} K(a_L(x) - a_L(x_i)) \tag{5.9}$$

$$LSA(x) = -\log(f(x)) \tag{5.10}$$

$a_L(\cdot)$ denotes a vector of activation values in the selected layer L. $\mathbf{X}$ denotes the training data (i.e., valid data). $f(x)$ is the density function estimated by KDE (denoted as K), where a higher density indicates the input x is similar to the training data. A larger $LSA(x)$ indicates more surprise of the input x.

Distance-based Surprise Adequacy (DSA) [15] uses Euclidean distance between the activation traces of a new input x and the training data as the measure of surprise. In this paper, we define the reference point x_n to be the nearest neighbor of x in the training data and compute DSA as follows.

$$x_n = \underset{x_i \in \mathbf{X}}{\operatorname{argmin}} \|a_L(x) - a_L(x_i)\| \tag{5.11}$$

$$DSA(x) = \|a_L(x) - a_L(x_n)\| \tag{5.12}$$

We use min-max normalization to rescale the LSA and DSA values for a better illustration across settings. Larger values indicate a higher surprise. We used activation traces output by the fourth last hidden layer for both the MNIST and Udacity datasets with a dimensionality of 64 and 1164, respectively.

DeepGini [13] estimates the uncertainty of the DNN classification, which is based on the test inputs activations of the DNN model's softmax output layer. It measures the likelihood of misclassification. It is defined as:

$$DG(x) = 1 - \sum_{c=1}^{C} p_c(x)^2 \tag{5.13}$$

With input x, a DNN model outputs a classification probability $p_c(x)$ for class c. With a completely certain classification (i.e., the model assigns a probability of 1 to the predicted class and 0 to others), we have $DG(x) = 0$. The DeepGini score increases with the classification uncertainty. The dependency on the softmax layer also restricts it to classification problems.

MC-Dropout measures uncertainty on regression problems, we compute the *Standard Deviation (std)* [131] induced by Monte-Carlo Dropout during prediction. Specifically, we compute the standard deviation of $N = 100$ sampled point prediction results for each test input x:

$$std = std([m(x)_1, ..., m(x)_N]), \tag{5.14}$$

where m signifies the model with dropout enabled.

Observations. Table 5.3 reports the average score of each metric for valid and invalid inputs, respectively. Note that in some cases, results are unavailable due to no invalid or valid inputs generated by the corresponding TIG. To conduct a fair comparison, our analysis excludes those situations.

Across two datasets, for both NC and KMNC, 83.3% of the time coverage achieved by valid inputs is greater than that achieved by invalid inputs.

In terms of SA, we expect invalid inputs can bring more surprise to the DNN model. 66.6% of the time invalid inputs achieve higher LSA than that of valid inputs across two datasets. 100% of the time invalid inputs achieve higher or the same DSA, which confirms a positive relationship. As for uncertainty-based metrics, we expect invalid inputs can induce higher uncertainty when the model makes predictions. For DeepGini, in MNIST, 100% of the time, invalid inputs induce higher uncertainty. Specifically, the score for invalid inputs is 0.15 higher than that of valid inputs on average. For Std in Udacity, the standard deviation of model predictions for invalid inputs is 0.04 higher than that for valid inputs.

Finding 4: Invalid inputs consistently (100% of the time) inflate uncertainty-based metrics. DSA reflects input invalidity more effectively than LSA. In 83.3% cases, valid inputs achieve higher NC/KMNC.

5.2 Our Approach

Based on our findings, we enhance existing white-box testing techniques to generate valid inputs for DNN testing by incorporating distribution awareness, which prevents

Table 5.3 Results for Test Selection Metrics. Val. and inval. denote results with valid inputs and invalid inputs, respectively. Tot. and Avg. denote total and average results for the entire test suite.

Dataset	TIG	NC (%)			KMNC (%)		
		val.	inval.	tot.	val.	inval.	tot.
MNIST	DeepXplore-occl	**72.6**	34.6	73.2	**35.7**	3.77	36.4
	DeepXplore-light	**84.3**	62.1	85.0	**41.8**	1.00	42.0
	DeepXplore-black	**79.2**	60.3	80.2	**36.6**	14.9	40.3
	DLFuzz	**71.0**	45.7	72.0	**37.7**	7.02	39.3
	Sinvad	62.2	**66.0**	70.2	27.9	**28.5**	40.9
Udacity	DeepXplore-occl	77.5	-	77.5	19.8	-	19.8
	DeepXplore-light	**85.7**	53.5	86.2	**17.8**	8.00	21.9
	DeepXplore-black	81.4	-	81.4	19.5	-	19.5
	DLFuzz	71.3	-	71.3	17.6	-	17.6
	Sinvad	-	80.1	80.1	-	20.4	20.4

(a) Results for NC and KMNC

		LSA			DSA			DeepGini / Std		
		val.	inval.	avg.	val.	inval.	avg.	val.	inval.	avg.
MNIST	DeepXplore-occl	0.45	**0.47**	0.45	0.47	**0.48**	0.47	0.35	**0.46**	0.35
	DeepXplore-light	**0.11**	0.05	0.11	0.58	**0.72**	0.58	0.55	**0.90**	0.55
	DeepXplore-black	**0.22**	0.19	0.21	0.39	**0.44**	0.40	0.31	**0.40**	0.33
	DLFuzz	0.37	**0.51**	0.38	0.50	**0.63**	0.51	0.16	**0.36**	0.18
	Sinvad	0.30	**0.34**	0.32	0.38	**0.46**	0.42	0.51	**0.53**	0.52
Udacity	DeepXplore-occl	0.09	-	0.09	0.24	-	0.24	0.07	-	0.07
	DeepXplore-light	0.30	**0.38**	0.32	0.48	0.48	0.48	0.05	**0.09**	0.06
	DeepXplore-black	0.14	-	0.14	0.33	-	0.33	0.04	-	0.04
	DLFuzz	0.35	-	0.35	0.40	-	0.40	0.05	-	0.05
	Sinvad	-	0.29	0.29	-	0.40	0.40	-	0.07	0.07

(b) Results for surprise-based and uncertainty-based metrics

Algorithm 4 Enhanced Testing Framework by Joint Optimization.

Require: X - Seed inputs
Require: obj_1 - Original objectives of TIG
Require: obj_2 - Distribution-aware objectives of IV
Require: DNN - DNN under test
Require: hyperparameters - $\{lr, \gamma, t, iters\}$
1: $test_suite = \{\}$
2: **for** x **in** X **do**
3: **for** i=1 **to** iters **do**
4: $obj = obj1 + \gamma \times obj2$
5: $G = \partial obj / \partial x$
6: $G = Process(G)$
7: $x = x + lr \times G$
8: **if** Misbehavior(DNN,x) and PassValidityCheck(x,t) **then**
9: $test_suite.append(x)$
10: break
11: **end if**
12: **end for**
13: **end for**

the generated input from deviating significantly from the learned distribution. We examine the quality of generated test cases by both humans and automated IVs. With **the same** 100 seed inputs, the results show that our enhanced framework can generate up to 10% more valid test inputs.

To introduce distribution awareness, we perform joint optimization, in which the best IV's objective is optimized together with each TIG's objective (e.g., DAIV for DeepXplore-blackout for MNIST, as shown in Table 5.2).

We also validate the effectiveness of TIG's original objective and only keep those that assist in generating valid inputs. For example, we keep the DeepXplore's NC objective according to *Finding 4: valid input achieves a higher coverage for all TIGs except SV.* It should be emphasized that this improvement is specifically applicable to white-box testing techniques that employ gradient-based optimization, and it does not extend to black-box techniques like SV, which relies on random feature perturbations.

Algorithm 4 elaborates details of the enhanced testing framework achieved through joint optimization. The algorithm requires five inputs outlined in the **Require** lines. Specifically, for X, we use the same 100 seed inputs as described in Section 5.1.1. For obj_2, we use probability density $\log p(x)$ for PCNN, reconstruction probability $p(x)$ for DAIV, negative feature distance $-d_k(x)$ between x and training data for D-KNN, and negative feature distance $-d_c(x)$ between x and the hypersphere center c

Table 5.4 Effectiveness of Joint Optimization (% valid). The positive/negative/equal signs indicate the improved techniques produce more/fewer/the same number of valid test inputs compared to the original techniques with the same seed inputs.

Data	TIG	DAIV	D-SVDD	D-KNN	PCNN	Human
MN	DO+KNN	3(+)	30(-)	100(+)	30(+)	98(+)
	DL+SVDD	83(-)	100(+)	80(+)	83(-)	99(=)
	DB+DAIV	100(=)	96(+)	62(+)	100(+)	83(+)
	DF+KNN	44(+)	98(+)	100(+)	5(+)	95(+)
UD	DO+SVDD	93(-)	100(=)	96(-)	78(+)	100(=)
	DL+SVDD	84(-)	100(+)	85(+)	93(=)	85(+)
	DB+SVDD	82(+)	100(=)	94(-)	91(+)	100(=)
	DF+SVDD	80(+)	100(=)	100(+)	0(=)	100(=)

for D-SVDD. For hyperparameters, we set $\gamma = 0.01$ through grid searching candidate values $[0.001, 0.01, 0.05, 0.1]$. We determine t as described in Section 5.1.3. lr and $iters$ values are set as in the original TIG works. We begin by initializing the *test_suite* to store the generated test inputs (Line 1). The generation process starts with looping over each seed image x in X (Line 2-3). We construct a joint objective obj by the weighted sum of $obj1$ and $obj2$ (Line 4). We maximize the joint objective and generate a new test input by performing gradient ascent. Note that the gradients G are processed by domain constraints, if any. (Line 5-7). We add the generated input to the test suite if it is confirmed valid by the selected IV (ValidityCheck) and causes model misbehavior (Line 8-10).

5.3 Results

5.3.1 Validity Assessments

The validity assessment results are presented in Table 5.4. Note that all test inputs selected with the chosen IV are 100% valid due to the ValidityCheck of our method. Comparing these results to Table 5.1, we observe that 78% of the time, automated IVs agree that joint optimization is beneficial in generating more or the same number of valid test inputs. It is also crucial to consider IV's reliability when they do not acknowledge the improvement. Human assessors consistently provide positive feedback (100% of the time), in which we observe a 2% to 10% increase in the number of valid test inputs.

Table 5.5 Improvement of DNNs Accuracy with "Random", "Original", and "Enhanced".

MNIST		Udacity	
Random	99.55%	Random	99.37%
Original	99.56%	Original	99.55%
Enhanced	99.61%	Enhanced	99.60%

5.3.2 Improving DNNs with Enhanced Framework

To analyze the implications on DNN's performance of the generated inputs, for both datasets, we augment the training data by adding 100 newly generated misbehavior-inducing inputs and further train the DNNs with five additional epochs [4, 15]. Specifically, we compare three augmentations: (1) "Random": 100 test inputs randomly selected from the original dataset; (2) "Original": 100 test inputs generated by the original testing framework; (3) "Enhanced": 100 test inputs generated by our enhanced testing framework. For MNIST, we adopt majority voting for labeling the test inputs generated with DeepXplore (following Pei *et al.* [4]), and adopt ground truth seed label for DLFuzz test inputs (we assume label preservation for minimal perturbations). For Udacity, we assume label preservation for all test inputs. Table 5.5 demonstrates the average performance over all testing techniques. The results show that our enhanced framework improves the accuracy of the test set by 0.05% to 0.23%. Note that we validate the label preservation assumption through expert review. For all minimally perturbed inputs, an expert confirmed that the perturbation was too minor to change the original label (e.g., the steering label in the Udacity dataset), even without explicit re-labeling.

5.3.3 Practicability

In our method, for each TIG, we select the best IV for joint optimization based on our pre-study results on 100 seed inputs. To apply the enhanced framework in practice, we suggest two ways: the first would be to reduce the number of seed inputs for pre-study (i.e., selecting only 10 samples can obtain an initial idea). The second would be directly referring *Finding 3*, which suggests applying feature-based IVs for both datasets. We plan to experimentally validate this assumption on more datasets in our future work.

Result: Domain experts have consistently (100% of the time) confirmed the effectiveness of our enhanced framework in valid input generation, demonstrating an up to 10% increase in the number of valid inputs.

5.4 Threats to Validity

5.4.1 Internal Validity

This can be attributed to the implementation quality of the existing techniques (e.g., TIGs, IVs, test selection metrics) and the reproducibility of our method. To mitigate this effect, we have open-sourced our implementation for transparency. Also, we ensure the same functionality of DNN models when converting TensorFlow-Keras to the PyTorch framework by using the same architecture and copying trained weights.

5.4.2 External Validity

A possible threat to the generalization of our experiment conclusions can stem from the choice of datasets, DNN models and studied problems. We try to alleviate the threat as follows: (1) We study both classification and regression tasks. The handwritten digit classification and autonomous driving steering prediction problems are representative and important in the DL research. (2) We choose different complexities of datasets, from 28×28 gray-scale MNIST to 640×480 RGB Udacity dataset. (3) We select DNN models that are widely used in the literature and reach competitive performance.

5.5 Chapter Summary

In this chapter, we undertake an empirical study and enhance the testing framework by incorporating distribution awareness. Our investigation reveals that the current TIGs can generate invalid inputs, and the average accuracy of automated IVs ranges from 49% to 77%. Furthermore, we analyze the relationship between six test selection metrics and input validity, discovering that invalid inputs increase the value of uncertainty-based metrics in 100% cases and decrease the value of coverage-based metrics in 83.3% cases.

Based on these findings, we improve the current testing framework by integrating IVs' distribution-aware objectives to generate more valid inputs. The results demonstrate a 2% to 10% increase in the number of valid inputs, as assessed by human

evaluation. In future work, we aim to extend our approach to other fields to enhance the completeness and understand the generalizability of our findings.

Chapter 6

Unified Ensemble-Based Testing for Fault Detection and Retraining in Deep Neural Networks

Testing deep learning (DL)-based systems is critical for identifying internal defects and improving their robustness. Existing test selection metrics have shown promising results and focus on individual aspects like uncertainty or neuron coverage, but fail to consider consensus decisions across metrics, limiting fault detection and retraining effectiveness. Specifically, we observed three key limitations: 1. Lack of consideration for the consensus of results from multiple independent metrics (i.e., selecting inputs based on a fused score from multiple metrics); 2. Insufficient effectiveness evaluation; 3. Limited scope of tasks and datasets.

In this chapter, we propose an ensemble-based testing framework that integrates scale and granularity unification to systematically combine multiple test selection metrics. We examine the effectiveness of test selection metrics on both the capability of fault detection and the capability of retraining guidance, and assess the performance on two different domains. Experimented and validated on two Android malware and two real-world driving datasets, our method achieves statistically significant improvements in fault detection (e.g., detecting 23 faults on average) and retraining guidance (e.g., achieving an average of 34.04% accuracy improvement for malware datasets and 13.22% for driving datasets). Compared to state-of-the-art baselines, our framework offers superior testing reliability and robustness across diverse out-of-distribution (OOD) scenarios. Additionally, our ablation study statistically validates the effectiveness of each component of our method, including scale

unification and ensemble-based selection. All code and data are open-source to support further research. This work was submitted to IEEE Transactions on Software Engineering in 2025.

The subsequent sections of this chapter are organized as follows: Section 6.1 illustrates the study design, including a comprehensive test suite evaluation, the design of our empirical study, our proposed ensemble-based testing framework, and the study overview. Section 6.2 details our experimental setup, containing studied test selection metrics, studied datasets and models, OOD operators and selection budgets, hyperparameters, and research questions. Section 6.3 illustrates the experimental results. Section 6.4 presents threats to validity, while Section 6.5 summarizes this chapter.

6.1 Study Design

Our study consists of three parts: (1) a comprehensive test suite evaluation, (2) an empirical study on clustering pipelines and retraining processes. Then, we assess the effectiveness evaluation of five existing test selection metrics, and (3) the development of the proposed ensemble-based testing framework. The first part establishes two evaluation criteria for test selection metrics, addressing the limitation found in the current literature. The empirical study evaluates five existing selection metrics based on these criteria, identifying the best-performing ones. Finally, our proposed ensemble-based testing framework leverages the identified metrics to perform enhanced testing.

To comprehensively compare test selection metrics, we evaluate their effectiveness across two key aspects: test input prioritization for DNN fault detection and retraining guidance for DNN performance improvement. The former aims to prioritize test inputs that are more likely to trigger diverse model misbehavior, enabling fast detection of DNN faults. Regarding retraining guidance, test selection metrics are used to select test inputs that are insufficiently represented during training for future enhancement of model performance through retraining. Metrics that can guide more effective retraining with a limited number of inputs are preferred. Note that these two capabilities assess two different aspects of a metric: pick an input that leads to a DNN fault and pick an input that is insufficiently represented during training. These two criteria are used to compare different test selection metrics throughout the paper.

6.1.1 Comprehensive Test Suite Evaluation

Existing literature adopts either insufficient or potentially biased evaluation criteria on the selected test suite. Our study mitigates the issues and comprehensively evaluates studied metrics through two criteria: the capability of fault detection and the capability of retraining guidance.

Capability of Fault Detection It is essential for a metric to effectively select a test suite that reveals diverse causes of model misbehaviors. We denote each root cause of misbehavior as a fault in the DNN under test [163]. Previous works have simply assessed the misprediction-detection rate [13, 81], which is insufficient when many test inputs are typically mispredicted for the same reason [88]. As stated in [164], *"selecting or generating test inputs that trigger misbehaviors is far more useful when these misbehaviors are diverse."*. Therefore, it is more effective to detect mispredictions with distinct causes. Inspired by Attaoui *et al.* [163] and Aghababaeyan *et al.* [16], we leverage a clustering-based approach to group similar mispredicted inputs, aiming for each cluster to represent a common set of features that are the plausible cause for mispredictions, i.e., high-quality cluster could correspond to a fault (empirically validated in Sec. 6.3.2). This form of clustering-based root-cause analysis has been shown to be effective across various data modalities, including but not limited to image data [163, 16, 165] and time series data [**? ?**]. Therefore, we rely on the clustering-based fault detection approach for both data types, containing three core steps: feature extraction, dimensionality reduction, and clustering. Algorithm 5 provides details of the clustering pipeline, with lines 2 to 4 outlining the three steps. Line 5 evaluates the quality of the resulting clusters.

Algorithm 5 The clustering pipeline.

Require: (X_s, y_s): the selected test suite and its ground truth labels
Require: $\hat{y}_s$: predicted labels of X_s by the model
Require: FE: Feature Extractor
Require: DR: Dimensionality Reduction Technique
Require: CA: Clustering Algorithm
1: Get misprediction set $X_s^{mis} \subseteq X_s$ where $\hat{y}_s \neq y_s$
2: Extract Features $F = FE(X_s^{mis})$
3: Reduce Dimensionality $F' = DR(F)$
4: Clustering $\{C_1, ..., C_K\} = CA(F')$
5: $Silhouette, DBCV = (F', \{C_1, ..., C_K\})$
6: **return** $\{C_1, ..., C_K\}, Silhouette, DBCV$

Feature Extraction: We adopt different feature extractors based on the data type studied: RGB image-based driving data and Android malware data.

For driving data, we leverage transfer learning, an efficient method in transferring knowledge between domains for image data [166]. Building on the findings of Attaoui *et al.*, who validated clustering-based fault detection pipelines for image-processing DNNs, we adopt their recommended pipeline: ResNet-50 (penultimate layer) for feature extraction, UMAP for dimensionality reduction, and DBSCAN for clustering. This setup is particularly suited to the Dave2-based steering prediction model used in real-world driving tasks.

For malware data, we do not rely on transfer learning due to the lack of established pretrained models. Instead, we extract features from our own trained detectors on the AndroZoo and Drebin datasets. We considered two candidate extractors, DeepDrebin and BasicDNN. We adopt shallow-layer features which better preserve input-specific details [55] and also achieve higher Silhouette and DBCV scores than deep-layer features in our preliminary experiments.

Dimensionality Reduction: We evaluate three widely used dimensionality reduction techniques: Principal Component Analysis (PCA) [167], Uniform Manifold Approximation and Projection (UMAP) [168], and Gaussian Random Projection (GRP) [169]. PCA is a linear method that transforms data onto a new coordinate system such that the principal components capturing the largest variation in the data can be easily identified. The core hyperparameter in PCA is the number of components. UMAP is a non-linear technique that preserves both local and global data structure by embedding high-dimensional data into a low-dimensional manifold. The key hyperparameters of UMAP include the number of components m indicating the dimensionality of the reduced features, the minimum distance d controls how tightly UMAP is allowed to pack points together, and the number of neighbors k that balances local versus global structure in the data (e.g., low values will force UMAP to concentrate on very local structure). GRP projects high-dimensional data onto a lower-dimensional subspace using a random matrix with entries drawn from a Gaussian distribution, approximately preserving pairwise distances.

To find the optimal hyperparmeter value, for each method, we perform a grid search on a range of candidate values and select the best value using the average of Silhouette and DBCV scores. The scores are computed from the resulting clusters with DeepDrebin as the feature extractor and DBSCAN as the clustering algorithm.

Clustering: We evaluate four clustering algorithms: K-Means [170], Hierarchical Agglomerative Clustering (HAC) [171], Density-based spatial clustering of applications with noise (DBSCAN) [172], and Hierarchical DBSCAN (HDBSCAN) [173]. K-Means divides the data into the required number of clusters based on

the distance calculated from the data points to the center of the cluster. HAC is a bottom-up clustering method that recursively merges the closest pairs of clusters based on a linkage criterion to build a tree-like hierarchy of nested clusters. Both K-Means and HAC require a pre-defined number of clusters K, where we use the knee-point method to identify the optimal value. We follow the same procedure [165] to apply the knee-point approach to the sum of distances of each point to its cluster center.

DBSCAN groups dense regions and treats low-density points as outliers, while HDBSCAN extends this by constructing a hierarchy to capture clusters of varying densities. DBSCAN requires two hyperparameters: the distance threshold ϵ and the minimum number of neighbors Min_k. HDBSCAN addresses the limitation of setting a global ϵ and only needs a pre-defined minimum cluster size $MinPts$. We identify the optimal ϵ value using the knee-point method, which first computes the Euclidean distance from each feature to its closest neighbor, then finds the knee-point of the curve obtained by considering those distances in ascending order [165]. To select an optimal Min_k and $MinPts$, we execute multiple candidate values with the best ϵ value and select the configuration that corresponds to the highest Silhouette and DBCV scores.

Evaluation: To identify the optimal clustering pipeline for malware data (note that we adopt the best pipeline suggested by [165] for driving data), we perform an ablation study on each component while keeping the others fixed. For a fair comparison, all techniques are evaluated using their optimal hyperparameters (see Table 6.3 in Section 6.2.5). Specifically, to compare feature extractors, we fix UMAP for dimensionality reduction and DBSCAN for clustering. UMAP is chosen due to its demonstrated effectiveness in preserving data structure and enhancing clustering performance [174, 175, 165]. We use density-based clustering methods, as they naturally handle arbitrarily shaped clusters better than K-Means or HAC, and select DBSCAN over HDBSCAN based on empirical results in [165]. After determining the best extractor, we evaluate dimensionality reduction techniques with the optimal extractor and DBSCAN. Each pipeline is assessed using the Silhouette [176] and DBCV scores [177] on the resulting clusters. These are widely used standard metrics for evaluating the quality of clustering [16]. Both scores range from -1 to 1, with higher values indicating better clustering quality. The evaluation results for each method are detailed in Section 6.3.1. To examine whether each cluster represents a distinct fault, we have conducted qualitative and quantitative analyses in Section 6.3.2 to validate the cluster-to-fault correspondence by the optimal pipeline.

Capability of Retraining Guidance. Retraining with test inputs that are under-represented or more likely to be misclassified is a straightforward way to

improve model performance. The performance improvement achieved after retraining with the selected test suite also reflects a metric's capability to identify the inputs that were insufficiently learned during the training process.

The approach to conducting the retraining process is crucial for enhancing DNN performance. Two types of retraining [1] have been explored in the existing literature [18, 13, 81, 86]: (1) Type I: retraining using the selected test suite only; (2) Type II: retraining using the combination of the original training set and the selected test suite. Algorithm 6 illustrates the process of both retraining types. Retraining is performed on a pre-trained model f with weight parameters θ, trained on the original training dataset. Given a selected test suite (X_s, y_s), selected from the hybrid candidate set (X_c, y_c) (we use the term *hybrid* to refer to a set that contains both ID and OOD data. Details on how the hybrid set was constructed can be found in Section 6.2.3), type I requires only (X_s, y_s) for retraining, whereas type II needs both (X_s, y_s) and the original training set (X_{tr}, y_{tr}). The performance of the retrained model is evaluated on the hybrid test set (X_t, y_t), which follows the same ID+OOD distribution as the hybrid candidate set [15, 86, 18]. In line 1, we first compute the prediction accuracy Acc_{ori} of the original pre-trained model $f(\theta)$ on the hybrid test set (X_t, y_t). If using type I retraining, only the selected data is used for retraining; if using type II retraining, both the training data and the selected data are used (lines 2-6). Then, we use mini-batch gradient descent with the cross-entropy loss L to retrain the model for a specified number of epochs (lines 7-12). After retraining, we compute the updated prediction accuracy Acc_{new} on (X_t, y_t) and measure the corresponding accuracy improvement Acc' to evaluate the effectiveness of retraining (lines 13-15).

6.1.2 Empirical Study

We conduct an empirical study to comprehensively evaluate existing test selection metrics using two distinct evaluation criteria. The best-performing metrics will be integrated into our proposed testing framework in the next section.

This section consists of two parts: the first part focuses on identifying the optimal clustering pipeline and retraining process to effectively capture the capability of fault detection and the capability of retraining guidance, respectively. For the former, we evaluate several commonly used methods at each step of the clustering pipeline (Algorithm 5). For the latter, we evaluate two types of retraining processes (Algorithm 6): type I and type II.

[1] We use the term *retraining* for consistency with previous studies [18, 4, 13, 16, 81].

Algorithm 6 Capability of retraining guidance.

Require: $f(\theta)$: the DNN model under test with pre-trained parameters θ
Require: (X_s, y_s): the selected test suite and its ground truth labels
Require: (X_{tr}, y_{tr}): the original train set
Require: (X_t, y_t): the hybrid test set and its ground truth labels
Require: L: the loss function
Require: hyperparameters - $\{epochs, bs, lr\}$
$Acc_{ori} = mean(f(X_t; \theta) == y_t)$
if Type I **then**
    X, y = X_s, y_s
else if Type II **then**
    X, y = concat(X_{tr}, X_s), concat(y_{tr}, y_s)
end if
for i in epochs **do**
    Shuffle(X, y)
    for X_b, y_b in (X, y) **do**
        $\theta \leftarrow \theta - lr \times \nabla_\theta L(f(X_b; \theta), y_b)$
    end for
end for
$Acc_{new} = mean(f(X_t; \theta) == y_t)$
Acc' = $Acc_{new} - Acc_{ori}$
return Acc'

The second part of our study evaluates the effectiveness of the selected metrics in terms of two aforementioned capabilities. For fault detection, we report the number of clusters identified, where each cluster is treated as a distinct model fault. The clusters are obtained using the optimal pipeline. For retraining guidance, we adopt the most effective retraining process identified and present the accuracy improvements as results.

6.1.3 Ensemble-based Testing Framework

Unification of Granularities and Scales. Existing test selection metrics derive scores either from individual test inputs or from a collection of test inputs (i.e., a test suite). Based on these two distinct approaches, we categorize them into two types in this paper: input-specific metrics (e.g., DeepGini, Entropy) and suite-wise metrics (e.g., geometric diversity, neuron coverage). The difference in measurement granularity between these two categories makes it challenging to directly combine them. Additionally, metrics within the same category also operate on a different scale, making it impractical to fuse their scores without adjustment. Scores can be transformed into different forms, for example, to a normalized value or even to a probability value. Our approach solves different scales within each category and different granularity across categories correspondingly by (1) using p-value/min-max to unify different scales of the score for input-specific/suite-wise metrics and (2) evaluating the rescaled suite-wise score on the refined set of test inputs, filtered by the rescaled input-specific score.

Input-specific Metrics. For metrics that compute scores based on individual test cases (e.g., DeepGini, Entropy, Surprise Adequacy), we leverage p-values for score unification.

Scale Unification: We assume scores following an empirical distribution constructed from the scores $S(X)$ of training data. Then, for each test input x, we transform its score $S(x)$ (e.g., DeepGini score) into probability values. The p-value captures the prior probability of observing a value at least as extreme as $S(x)$ if the null hypothesis H_0 (i.e., $S(x)$ is drawn from the given distribution) were true:

$$\text{p-value}(x) = Pr(S(X) \geq S(x)|H_0) \tag{6.1}$$

The p-values range from 0 to 1, with smaller p-values indicating more extreme scores (e.g., higher uncertainty for the metric DeepGini).

Combination: To combine multiple p-values, we leverage the weighted sum of p-values for each input-specific metric m (i.e., meaningful in computing score on

individual inputs), each with weight w_m. Let M_{ind} denote the number of input-specific metrics we would like to combine.

$$S_{ind}^{x} = \text{Combined p-value}(x) = \sum_{m=1}^{M_{ind}} w_m \times \text{p-value}(x)_m \tag{6.2}$$

Suite-wise Metrics. Metrics applied to the entire test suite (e.g., Geometric Diversity, Neuron Coverage, Surprise Coverage) is termed as suite-wise metrics in this paper. We first explain the process of achieving granularity unification with input-specific metrics. Subsequently, we detail the scale unification across different suite-wise metrics.

Granularity Unification: To integrate input-specific metrics with suite-wise metrics, we compute the suite-wise score based on a filtered test suite, guided by input-specific metrics. The process begins by filtering initial candidate inputs using input-specific metrics to create a refined candidate pool. The filtering process is expected to (1) improve the quality of the candidate pool by prioritizing high-risk inputs, and (2) enhance the efficiency of selection by reducing the search space. Then, we randomly construct several candidate test suites from the filtered candidate pool and select the one with the highest suite-wise score. Taking the metric GD as an example, we select the test suite that has the highest geometric diversity from the candidate test suites.

Scale Unification: If we want to combine multiple suite-wise scores, we use the relative scaling for unification. In this case, we opt for the min-max normalization. We do not use p-values since it can be extremely computationally expensive for suite-wise metrics. Specifically, p-values are computed from the empirical training distribution composed of scores of all possible test suites, and there can be a potentially infinite number of test suites with a large selection pool (e.g., there are 5E+71 ways to select 40 samples only from 1000 samples). Without the need to determine the underlying distribution of the scores (e.g., z-score assumes a roughly normal distribution), we adopt the distribution-agnostic min-max normalization. This approach preserves the relational properties of the data while offering computational efficiency [178].

Combination: Let $S(X_s)$ be the score computed from test suite X_s, and $max(S)$ and $min(S)$ be the maximum and minimum possible score of the metric of interest:

$$S_{st} = \sum_{m=1}^{M_{st}} w_m \times \frac{S(X_s) - min(S)}{max(S) - min(S)} \tag{6.3}$$

where w_m denotes the weight for the suite-wise metric m. The min-max normalized score ranges from 0 to 1.

Overview. In this section, we provide an overview of our proposed ensemble-based testing framework. The algorithm details are summarized in Algorithm 7, including Input, Test Input Selection process, and Output.

Input: The algorithm requires four key inputs outlined in the **Require** lines. The hybrid candidate set (X_c, y_c), containing a set of test inputs for selection. y_c indicates the ground truth label set. r represents the portion of samples to retain after the filtering process. N indicates the total number of samples to select, which depends on the selection budget. Note that the number of filtered samples is required to be larger than N. n is the number of random selections.

Test Input Selection: Firstly, given the interested input-specific metrics, we iteratively compute the input-specific score S^x_{ind} (Equation 6.2) for each input sample x (lines 1-3). Then, we sort the candidate set (X_c, y_c) based on the score and obtain the sorted set (X_{sorted}, y_{sorted}) (line 4). Note that the sorting order depends on the score we used. For example, if we use uncertainty-based scores like Entropy, which recommends selecting the most uncertain samples. The sorting process will sort the scores from the largest to the smallest, where the largest represents the most uncertain sample. Line 5 computes the number of samples N_r to retain after filtering. $|\cdot|$ represents the cardinality of the set X_c, i.e., number of samples. Note that $N_r > N$. Then, we filter the candidate set by the given number N_r to reduce the search space, obtaining the filtered set $(X_{filtered}, y_{filtered})$ (lines 6-7). After that, we randomly select the test suite for n times, each with size N, and compute the suite-wise score S_{st} for these n suites (lines 8-11).

Output: After obtaining the suite-wise scores for each suite, the algorithm returns the best test suite (X^*_s, y^*_s) with the highest S^*_{st} (lines 12-13).

6.1.4 Study Overview

This section summarizes the overall workflow of our study. As shown in Figure 6.1, our work contains three parts: comprehensive test suite evaluation (*green*), empirical study (*yellow*), and our methodology (*purple*).

For the first part, we have visualized the aforementioned two evaluation criteria for a deeper understanding. The orange rectangle represents the selected test suite. Test inputs within the selected suite are either correctly predicted (green dots) or incorrectly predicted (i.e., induce misbehavior, black dots) by the DNN under test. Each purple

Algorithm 7 Ensemble-based test input selection.

Require: (X_c, y_c): the hybrid candidate set
Require: r: ratio of samples to retain after filtering
Require: N: the number of samples to select, depending on the budget
Require: n: number of random selections
for (x, y) in (X_c, y_c) **do**
    Compute Input-specific score S_{ind}^{x}
end for
Sort (X_c, y_c) based on S_{ind}^{x}, obtain (X_{sorted}, y_{sorted})
$N_r = |X_c| \times r$
$X_{filtered} = X_{sorted}[: N_r]$
$y_{filtered} = y_{sorted}[: N_r]$
for i in n **do**
    Randomly selects (X_s, y_s) with size N from $(X_{filtered}, y_{filtered})$
    Compute suite-wise score S_{st} from X_s
end for
Select (X_s^*, y_s^*) with the highest S_{st}^*
return (X_s^*, y_s^*)

Table 6.1 Description of studied datasets, models, tasks, performance, and corresponding OOD operators.

Dataset	Description	#Train	#Test	Task (#Classes)	DNN	Val acc.	OOD inputs
AndroZoo [92]	Android packages	85,343	21,336	Classification 2 Classes	DeepDrebin BasicDNN	99.15 99.30	FGSM, GDKDE, Mimicry, Pointwise, Drebin, AndroZoo2018, 2019.
Drebin [2]	Android packages	38,245	9562	Classification 2 classes	DeepDrebin BasicDNN	99.50 99.53	FGSM, GDKDE, Mimicry, Pointwise, AndroZoo.
Dave [42]	Real-world driving video frames	27,243	9081	Classification 3 classes	Dave2V1 Dave2V2	97.99 98.04	FGSM, BIM, PGD, Momentum, Udacity.
Udacity [26]	Real-world driving video frames	101,396	5614	Classification 3 classes	Dave2V1 Dave2V2	99.52 99.49	FGSM, BIM, PGD, Momentum, Dave.

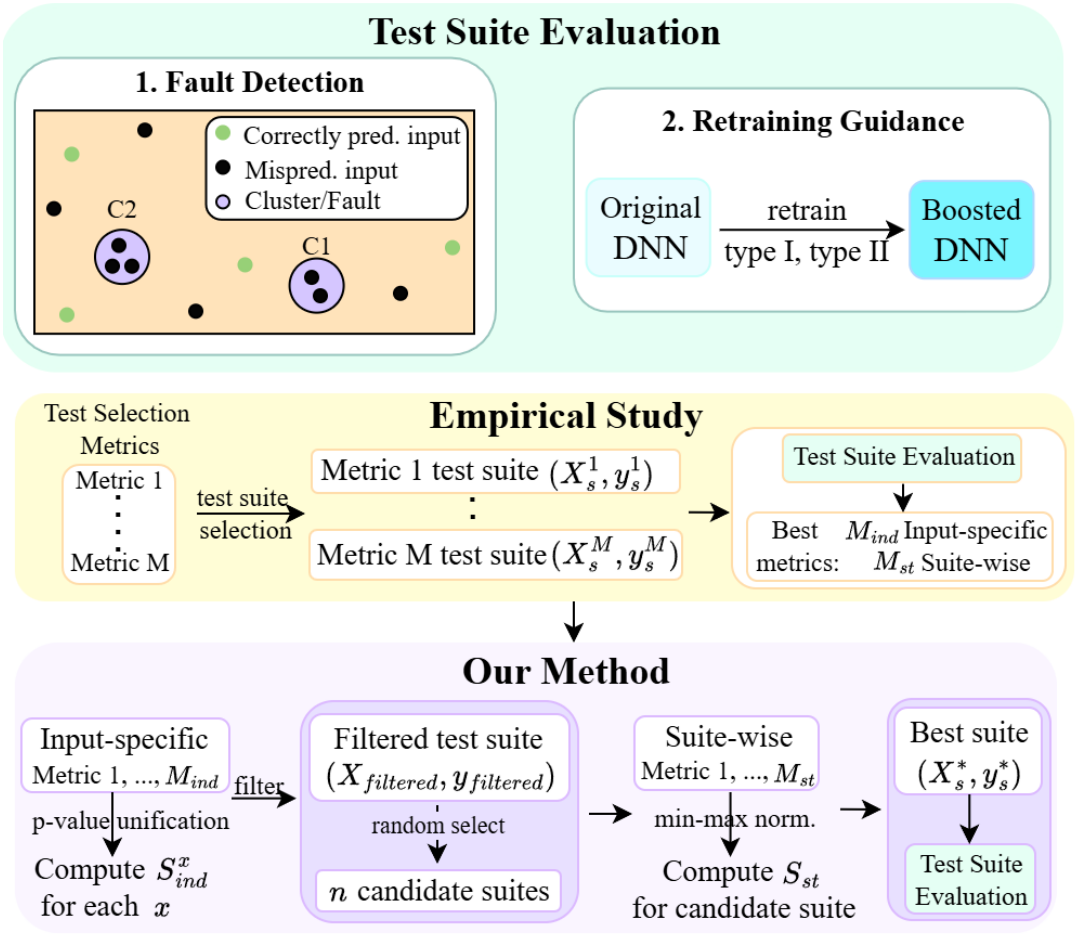


Fig. 6.1 Overview of our study. Part 1: More comprehensive test suite evaluation. Part 2: Empirical study. Part 3: Our methodology.

circle indicates a cluster, obtained by the clustering approach discussed in Section 6.1.1. Note that we only conduct clustering on the misprediction set (a set of black dots) since we expect each cluster to represent a common set of features that are the plausible cause for mispredictions. A larger number of clusters implies that more DNN faults are detected. Mispredicted inputs that do not belong to any cluster are regarded as noise and are excluded from our analysis, as we aim to obtain a reliable approximation of common faults. For the capability of retraining guidance, the dark blue DNN represents the model with enhanced weight parameters after retraining from the light blue DNN. We utilize the accuracy improvements as the evaluation metric, where higher improvements indicate better guidance in DNN retraining.

For the empirical study part, we have studied $M = 5$ test selection metrics in total (details elaborated in Section 6.2.1). The test suite selected by each metric is comprehensively evaluated through two criteria, as indicated in the green part. After the effectiveness evaluation, we identify several best-performing metrics, the number of pinpointed metrics is denoted as M_{ind} for input-specific metrics and M_{st} for suite-wise metrics, respectively. Then, we input them into our proposed methodology (see Section 6.1.3 for details).

The proposed framework consists of five steps. The first step is computing the combined score S^x_{ind} of each input x from input-specific metrics. Then, initial filtering is conducted to reduce input space for random selection, which is expected to avoid selecting redundant or less useful samples. After that, our method randomly selects n test suites and outputs the optimal suite (X^*_s, y^*_s) with the highest S^*_{st} (Equation 6.3).

6.2 Experimental Setup

6.2.1 Studied Metrics

We consider five test selection metrics for the empirical study.

Distribution-aware test selection (Dat) [18] (input-specific metric) is a hybrid test selection metric that selects uncertain samples from the ID data and performs random selection for the OOD data. They evaluated the effectiveness of DAT in both image and text domains.

DeepGini [13] (input-specific metric) is an uncertainty-based test selection metric that selects the most uncertain input samples based on the probabilities output by

models:

$$\underset{x \in X_c}{\text{argmax}}(1 - \sum_{i=1}^{C} p_i^2(x)), \tag{6.4}$$

with C being the number of classes, X_c being the candidate set, and $p_i(x)$ being the model prediction probability of class i with the input x.

Entropy [14] (input-specific metric) is also an uncertainty-based metric derived from the Shannon entropy from information theory. It selected the samples with maximum uncertainties:

$$\underset{x \in X_c}{\text{argmax}}(-\sum_{i=1}^{C} p_i(x) \log p_i(x)) \tag{6.5}$$

Geometric Diversity (GD) [16] (suite-wise metric) selects the test suite based on diversity. The diversity $G(X_s)$ of the selected test suite X_s is computed based on the extracted features:

$$G(X_s) = det(\phi(X_s) * \phi(X_s)^T), \tag{6.6}$$

where det indicates the determinant of a matrix.

Random: We also include random selection as a baseline metric, which selects the test input in a random manner. Each sample has an equal probability of being selected, regardless of the model behavior.

6.2.2 Datasets & Models

In this section, we elaborate on the details of the datasets (Drebin, AndroZoo, Udacity, and Dave) and models (DeepDrebin, BasicDNN, Dave2V1, and Dave2V2) studied in this paper. We have also summarized the description in Table 6.1. For the details of OOD operators, please refer to Section 6.2.3.

Datasets. We conduct experiments on four datasets across two domains, including two Android malware detection datasets and two autonomous driving image datasets.

AndroZoo [92] is an APK repository. We downloaded data that was collected in 2017, 2018, and 2019. For 2017, we directly utilize the features of 106,679 APK files extracted by Li *et al.* [179] (15,465 malware examples and 91,214 benign examples, from the date July 1st to December 31st in 2017), where the features can be directly used as the input to a malware detector. The data were classified based on the feedback from the VirusTotal service, which is an ensemble of over 70 anti-virus scanners (e.g.,

Kaspersky, McAfee, FireEye, Comodo, etc.). Specifically, the sample is labeled as malware if it is classified as malicious by at least five anti-virus scanners. The dataset was split into training (60%), validation (20%), and testing (20%). The data collected in the years 2018 and 2019 are used as the evaluation of temporal covariate shift (see Section 6.2.3 for details), which we downloaded from the official AndroZoo website [2].

Drebin dataset [2] contains 5615 malicious Android packages and SHA256 values of 123,453 benign samples. We downloaded features extracted by Li *et al.* [179], containing 42,328 benign APKs and 5479 malicious APKs.

Dave testing dataset [42] contains 45,568 real-world driving images. The images are recorded from a dashcam with labeled steering angles, captured around Rancho Palos Verdes and San Pedro California.

Udacity driving dataset [26] contains 101,396 training and 5614 testing RGB images captured by cameras mounted behind the windshield of a human-driving car. The steering angle applied by the human driver is also recorded along with each image frame. We use the competitive DAVE-2 self-driving car model from Nvidia [152] with dropout layers, achieving 99.1% accuracy[3].

We also provide examples of test inputs from these two domains. As illustrated in Figure 6.2, images in the top row are examples of test inputs (we use natural data here) from Udacity and Dave datasets. The bottom row shows examples of Android applications from AndroZoo and Drebin datasets, which will be transformed into binary feature vectors for DL-based models to process and learn.

Models. We have implemented four models in total, with two malware detectors and two driving models.

For malware detectors, we use two learning-based fully-connected models, namely BasicDNN and DeepDrebin, which are commonly used by the community [85, 179].

For the driving models, we have studied two CNN-based end-to-end steering models that achieved top performance on the leaderboard of Udacity self-driving car challenge 2 [180]: Dave2V1 and Dave2V2. Originally, the driving models output numerical continuous steering values with the input image data. To meet the requirements of most test selection metrics, we have transformed the regression models into classification models by discretizing steering angles using threshold values. Following [181], we change the final activation layer to a three-class softmax, indicating ‘turn left’, ‘go straight’, and ‘turn right’.

[2]https://androzoo.uni.lu/

[3]we report 1-MSE (Mean Squared Error) as the accuracy.

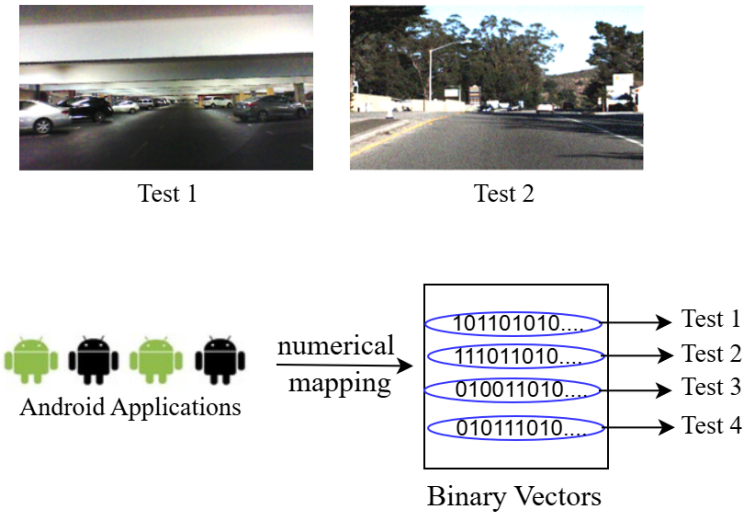


Fig. 6.2 Examples of test inputs from driving images (Udacity, Dave) and Android Packages (AndroZoo, Drebin).

Table 6.2 Description of studied OOD operators.

OOD Type	**Operator**	**Description**
Adversarial attack	FGSM [10]	Gradient-based white-box
	GDKDE [104]	Gradient-based white-box
	Mimicry [182]	Gradient-free grey-box
	Pointwise [183]	Gradient-free grey-box
	BIM [9]	Gradient-based white-box
	Momentum [105]	Gradient-based white-box
	PGD [184]	Gradient-based white-box
Out-of-source	Diff. dataset	Data with diff. dist.
Temporal covariate shift (AndroZoo only)	Diff. year	Data collected in a different time.

6.2.3 OOD Operators and selection budgets

We aim to decrease the computational and human effort to improve models when a distribution shift occurs. To simulate a generalized set of OOD samples, we introduce the distribution shift in three ways: (1) adversarial attacks, (2) out-of-source, and (3) temporal covariate shift. We denote samples with distribution shift as out-of-distribution (OOD) data and samples without distribution shift (e.g., training data) as in-distribution (ID) data. We summarize the description of each technique we employed to achieve the proposed distribution shifts in Table 6.2. Note that each dataset may adopt different operators.

In the context of adversarial attacks, we apply four attack techniques for each data type, selecting methods that are commonly used within their respective domains. For malware datasets (AndroZoo and Drebin), we employ FGSM, GDKDE, Mimicry, and Pointwise [85, 179]. For RGB driving data (Udacity and Dave), we employ FGSM, BIM, Momentum, and PGD [13, 28]. For out-of-source data, we leverage a different dataset that is not from the original distribution, we respectively use AndroZoo, Drebin, Udacity, and Dave for Drebin, AndroZoo, Dave, and Udacity. We only construct hybrid datasets with the temporal covariate shift for AndroZoo due to the limited availability of data samples taken from different times.

Hybrid set construction. For each OOD operator, we generate an OOD set from the original test set of each dataset. For example, we generate an adversarial sample by FGSM for each sample in the test set. For the out-of-source type, we create the OOD set by selecting the same number of samples from the out-of-source dataset. To construct the hybrid candidate set and the hybrid test set, we first randomly split the original test set into two equal-sized subsets with balanced labels. Then, we proportionally replace original ID samples with OOD samples to form each hybrid set. Note that we construct a hybrid set for each OOD operator. In this study, we examine eleven different ID+OOD distributions: 0%+100% (only contains OOD data), 10%+90%, 20%+80%, 30%+70%, 40%+60%, 50%+50%, 60%+40%, 70%+30%, 80%+20%, 90%+10%, 100%+0% (only contains ID data). After construction, the hybrid candidate set is used for test input selection by metrics, and the hybrid test set is used for retraining evaluation.

Selection Budgets. We select 1%, 3%, 5%, and 10% data from each hybrid candidate set based on each test selection metric. For the experiments comparing the two retraining processes (i.e., the first part of the empirical study), we also use 100% hybrid candidate data to retrain the model to exclude the effect of test selection metrics.

6.2.4 Research Questions

This section outlines research questions investigated in this paper. RQ1.1 and RQ1.2 empirically examine clustering pipelines for fault detection, while RQ1.3 evaluates retraining strategies. RQ2 evaluates existing test selection metrics through their capabilities of fault detection and retraining guidance. RQ3 studies our proposed test input selection framework.

RQ1.1. Assessment of different clustering pipelines. This research question investigates which combinations of feature extraction, dimensionality reduction, and clustering methods yield high-quality clusters for mispredicted samples. We evaluate several commonly used methods for each stage of the pipeline and assess clustering quality using Silhouette and DBCV scores.

RQ1.2. Validation of cluster-to-fault correspondence. This research question examines whether each cluster from the best clustering pipeline from RQ1.1 corresponds to a distinct root cause of model faults. We perform a two-part validation: (1) heatmap-based visualization to qualitatively assess intra-cluster feature coherence and inter-cluster feature dissimilarity, and (2) retraining-based validation to verify that retraining on test inputs results in greater improvement for samples within the same cluster compared to those from other clusters, providing empirical evidence that the clusters represent distinct faults.

RQ1.3 Comparison of different retraining strategies. To ensure a fair evaluation of test selection metrics, we compare two types of retraining strategies and identify the more effective one. While Hu *et al.* [18] previously showed that type II retraining outperforms type I in general, our study extends this analysis by (1) incorporating newly introduced test selection metrics, and (2) evaluating both strategies across two diverse data domains (malware and autonomous driving) not previously studied.

RQ2. Effectiveness of different metrics in two aspects. (1) Capability of fault detection; (2) Capability of retraining guidance. For (1), we demonstrate the number of faults of DNN misbehaviors identified by different metrics, using the best clustering pipeline from RQ1.1. For (2), we adopt the superior retraining process found in RQ1.3 and measure the accuracy improvements on hybrid test sets.

RQ3. Effectiveness of our proposed testing framework. In this research question, we evaluate the proposed test input selection process (Algorithm 7) across two criteria. Specifically, we complete our framework by incorporating the best-performing metrics, as identified in RQ2. For the fault detection criterion, we provide the number of clusters obtained from the best pipeline. For the retraining guidance criterion,

we provide the accuracy improvements by the superior process. In the first part of this research question, we conduct comparative analysis of our method with the best baselines along both criteria. In the second part, we compare our method with two variants to validate the effectiveness of each part of our method. We also conduct statistical analysis (Scott-Knott ESD test) to show the significance of the improvement in both parts.

In summary, RQ1.1 empirically examines different clustering pipelines. RQ1.2 validates whether each cluster corresponds to a distinct DNN fault. RQ1.3 evaluates different retraining strategies. Together, these establish that the adopted evaluation criterion can accurately reflect the targeted effect, ensuring a fair and meaningful comparison between metrics in subsequent analyses. RQ2 provides a comprehensive evaluation of existing test selection metrics along these two criteria. Building on the empirical findings from RQ2, RQ3 presents and validates the effectiveness of our complete test input selection framework.

Implementation. The implementation and data are available at https://github.com/EnsembleTesting/EnsembleTesting/. All experiments were performed on machines equipped with an Intel Xeon Silver 4210 CPU, and an Nvidia GeForce RTX 3090-Ti GPU.

6.2.5 Hyperparameters

Table 6.3 summarizes the hyperparameters considered in this study, including their descriptions, candidate values, selection methods, and the final optimal values adopted in our experiments.

To fairly compare methods in the clustering pipeline, we adopt optimal hyperparameter values. We perform clustering on mispredicted samples from the original test set for this preliminary analysis. For methods employing grid search, the optimal value is selected based on the average of the Silhouette and DBCV scores. For methods relying on the knee-point strategy, we provide detailed guidance on curve computation in Section 6.1.1. Regarding feature extraction, both DeepDrebin and BasicDNN contain two hidden layers. We experimentally observe that shallow-layer features consistently yield higher Silhouette and DBCV scores. For dimensionality reduction using UMAP, we define the candidate range of m such that $m \leq |X_s^{mis}|$, addressing the curse of dimensionality. This principle similarly applies to PCA and GRP. In practice, candidate values exceeding the number of mispredicted samples are filtered out. For parameters d and k, we follow the recommended ranges from

Table 6.3 Summary of hyperparameters, including their descriptions, candidate ranges, selection methods, and the final selected values.

Methods	Param.	Descriptions	Candidate values	Selection	Value
DeepDrebin	$layer$	Layer for	[dense1, dense2]	grid search	dense1
BasicDNN	$layer$	feature extraction	[dense1, dense2]	grid search	dense1
UMAP	m	No. of components	[5,10,25,50,75,100]	grid search	25
	d	Min. distance	[0.01,0.1,0.3,0.5]	grid search	0.1
	k	No. of neighbors	[5,15,25,50]	grid search	15
PCA	m	No. of components	[5,10,25,50,75,100]	grid search	25
GRP	m		[5,10,25,50,75,100]	grid search	25
DBSCAN	Min_k	Min. neighbors	[2,5,10,15]	grid search	2
	ϵ	Distance threshold	NA	knee-point	Varied
HDBSCAN	$MinPts$	Min. cluster size	[2,5,10,15]	grid search	2
HAC	K	No. of clusters	range[5,35]	knee-point	[10,10]
K-Means	K	No. of clusters	range[5,35]	knee-point	[9,15]
Retraining	bs	Batch size	[128,256,512]	grid search	128
	$epochs$	No. of epochs	[5,10,20,30,40]	grid search	30
	lr	Learning rate	[1e-2,1e-3,1e-4,1e-5]	grid search	[1e-3,1e-4]
Ours	w	Weight	[0,0.3,0.5,0.7,1.0]	grid search	0.5
	r	Ratio	[0.3,0.5,0.7,1.0]	grid search	0.7
	n	No. of selections	[30,60,120,180]	grid search	60

the official UMAP documentation.[4] The candidate values for Min_k and $MinPts$ in DBSCAN and HDBSCAN are determined by referring to official documentation[5],[6] and relevant prior work [16, 165]. For K-Means and HAC, we restrict the candidate range of K to [5, 35] to reflect the relatively small number of mispredicted inputs from the original test set. The selected optimal values are $K = 9$ for AndroZoo and $K = 15$ for Drebin in K-Means, and $K = 10$ for both datasets in HAC. Note that these values are used only for method comparison. Since the optimal ϵ varies across settings, we apply the knee-point method in each experiment to determine the best value. For the other aforementioned hyperparameters in the clustering pipeline, we find that the optimal values remain consistent across settings and thus fix them as shown in the table in our main experiments.

For both the retraining procedure and our proposed methodology, hyperparameter values are selected via grid search based on initial exploratory experiments. Specifically, we determine the optimal batch size (bs) and learning rate (lr) by identifying the configuration that yields the greatest accuracy improvement, while the number of training epochs is selected based on convergence behavior. We adopt a learning rate of $1e^{-3}$ for malware data and $1e^{-4}$ for driving data across both the retraining and original training processes. In our framework, we also tune parameters w (weight in metric combination in Eq. 6.2) and r (ratio of samples to retain after filtering in Alg. 7) from predefined candidate sets. For the number of random selections n in Alg. 7, we refer to prior work [16], and our preliminary experiments show that $n = 60$ achieves comparable performance to $n = 120$ and $n = 180$, while $n = 30$ leads to slight degradation. Therefore, we use $n = 60$ to balance effectiveness and computational efficiency.

6.3 Results

6.3.1 RQ1.1. Assessment of different clustering pipelines.

To effectively assess different clustering pipelines, we ablate the studied techniques in each step (i.e., feature extraction, dimensionality reduction, clustering) separately by fixing others. For example, to ablate feature extractors, we fix UMAP for

[4]https://umap-learn.readthedocs.io/en/latest/parameters.html
[5]https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html
[6]https://scikit-learn.org/stable/modules/generated/sklearn.cluster.HDBSCAN.html

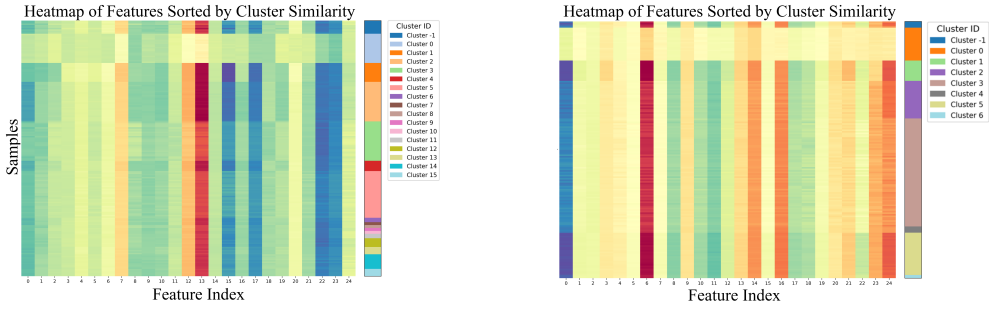
dimensionality reduction and DBSCAN for clustering. For details please refer to the evaluation part in Section 6.1.1.

Table 6.4 records the results of Silhouette and DBCV scores for each ablation experiment for malware data. Note that we did not present the number of clusters since it is misleading to compare them with the pipelines producing low clustering quality, i.e., low-quality clusters cannot be considered as the root causes of the faults [16]. Our results suggest that the pipeline relying on a DeepDrebin feature extractor, with UMAP and DBSCAN, is the best pipeline because it yields higher Silhouette and DBCV scores than other pipelines. Specifically, averaging over both malware datasets and models, the optimal pipeline yields a Silhouette score of 0.57 and a DBCV score of 0.78.

Table 6.4 Silhouette and DBCV scores for each candidate pipeline, average over all datasets and models. Larger scores indicate better clustering quality. Best values are in bold.

Component	Techniques	Silhouette	DBCV
Feature Extractor	DeepDrebin	**0.57**	**0.78**
	BasicDNN	0.42	0.32
Dimensionality Reduction	UMAP	**0.57**	**0.78**
	PCA	0.08	-0.40
	GRP	-0.01	-0.50
Clustering Algorithm	DBSCAN	**0.57**	**0.78**
	HDBSCAN	0.52	0.55
	HAC	0.19	0.20
	K-means	0.40	0.09

For driving data, we directly use ResNet-50 for feature extraction, UMAP for dimensionality reduction, and DBSCAN for clustering, as discussed in Section 6.1.1 and suggested by Attaoui *et al.* [165]. To validate the clustering quality, we also examine the Silhouette and DBCV scores on the best pipeline (ResNet-50, UMAP, DBSCAN) and low-performing pipeline (ResNet-50, PCA, K-Means) discovered by [165]. Averaged across all driving datasets and models, the best-performing pipeline achieves a Silhouette score of 0.49 and a DBCV score of 0.45. In contrast, the lower-performing pipeline yields a Silhouette score of 0.09 and a DBCV score of -0.32, indicating poor clustering quality.

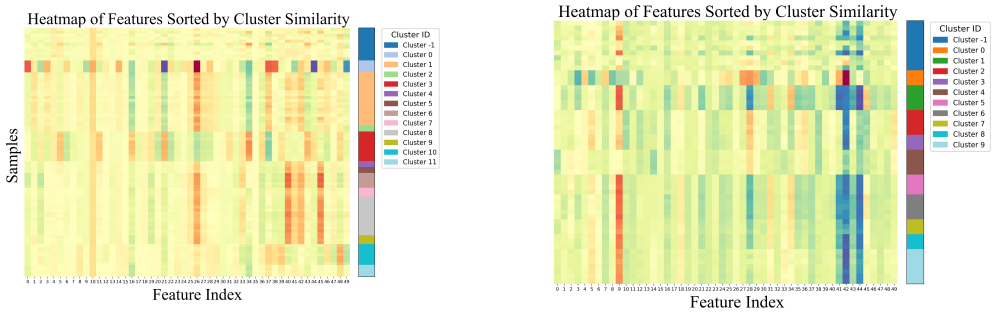


(a) BasicDNN (b) DeepDrebin

Fig. 6.3 Malware data: Heatmaps of high-quality clusters by the best pipeline (DeepDrebin, UMAP, DBSCAN). Visualizations of features of the misprediction set by BasicDNN and DeepDrebin, respectively. We also include noisy points as Cluster ID=-1 (top) for reference.

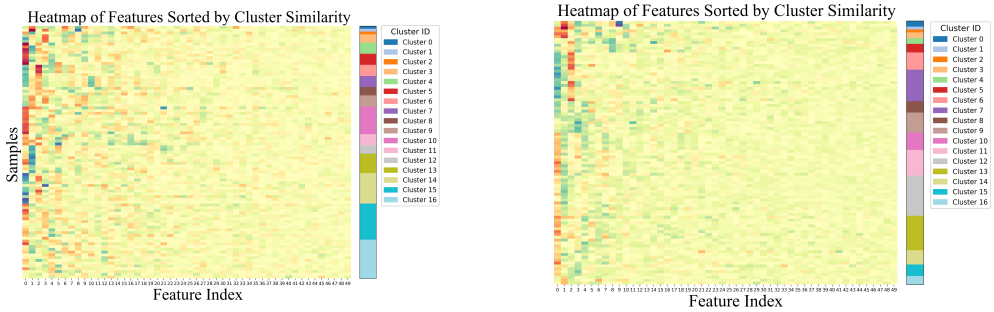
(a) BasicDNN (b) DeepDrebin

Fig. 6.4 Malware data: Heatmaps of low-quality clusters by another pipeline (DeepDrebin, PCA, K-Means). Visualizations of features of the misprediction set by BasicDNN and DeepDrebin, respectively.



(a) Dave2V1 (b) Dave2V2

Fig. 6.5 Driving data: Heatmaps of high-quality clusters by the best pipeline (ResNet-50, UMAP, DBSCAN). Visualizations of features of the misprediction set by Dave2V1 and Dave2V2, respectively. We also include noisy points as Cluster ID=-1 (top) for reference.



(a) Dave2V1

(b) Dave2V2

Fig. 6.6 Driving data: Heatmaps of low-quality clusters by another pipeline (ResNet-50, PCA, K-Means). Visualizations of features of the misprediction set by Dave2V1 and Dave2V2, respectively.

6.3.2 RQ1.2. Validation of cluster-to-fault correspondence.

We conduct qualitative and quantitative analyses to examine whether the high-quality clusters generated by the best-performing pipeline indeed correspond to distinct root causes of DNN faults.

For the **qualitative analysis**, we adopt heatmap visualizations to inspect the intra-cluster feature coherence and inter-cluster feature dissimilarities of mispredicted samples. Heatmaps offer a unified, color-coded matrix representation of feature values that is agnostic to the underlying data type, enabling consistent interpretation across heterogeneous datasets. We operate under the assumption that a root cause manifests as shared patterns or commonalities in the feature space. Therefore, clusters exhibiting high internal feature similarity are hypothesized to reflect the presence of a common underlying root cause, though the precise mechanism for this correspondence remains an open question. Specifically, for malware data, Fig. 6.3 and Fig. 6.4 provide heatmaps of high-quality (with the best-performing pipeline) and low-quality (another pipeline) clusters, respectively. The clustering pipeline is performed on the misprediction set of AndroZoo by DeepDrebin and BasicDNN models, respectively. We conduct the same analysis for driving data (Dave dataset by Dave2V1 and Dave2V2, as displayed in Fig. 6.5 and Fig. 6.6). Note that although we directly adopt the best pipeline recommended by [165], we visualize another low-performing pipeline (ResNet-50, PCA, K-Means) for comparison. Due to the large number of clusters generated by the misprediction set of driving data, we only show the first half clusters for simplicity. Note that the plots appear to be more crowded as the best dimensionality of the reduced feature is 50.

In each heatmap, rows represent individual samples (feature vectors) in the misprediction set, and columns represent feature elements. Feature values are color-coded using the Spectral colormap to highlight the inter-cluster differences and intra-cluster similarities. A vertical color bar on the right indicates cluster assignments (a random color is assigned to each cluster), including a distinct cluster label for noisy points (Cluster ID=-1). A legend identifies cluster IDs using corresponding colors.

By inspecting heatmaps of high-quality clusters (e.g., Fig. 6.3a, Fig. 6.3b, Fig. 6.5a, and Fig. 6.5b), the uniformity in feature patterns across rows within each cluster suggests high internal similarity, which may correspond to a shared underlying root cause of faults. To obtain a better understanding of dissimilarities across different clusters, we draw the heatmaps using sorted clusters by their similarities using centroid values. Specifically, Cluster 0 and Cluster 1 are the two most similar clusters. We can see that, even compared to the closest cluster, samples still demonstrate different feature colors, implying each cluster indeed corresponds to a unique root cause of the faults.

We also note that noisy points (Cluster ID = -1, placed at the top of heatmaps) do not exhibit feature patterns that are significantly different from the identified fault clusters, i.e., their color patterns are similar to at least one existing cluster. Moreover, the number of noisy points is relatively small compared to the total number of mispredicted samples. Therefore, we empirically conclude that the noisy points do not represent rare or dispersed faults and can be safely discarded [16].

In contrast, low-performing clustering pipelines (i.e., Fig. 6.4a, Fig. 6.4b, Fig. 6.6a, Fig. 6.6b) demonstrate more dispersed and inconsistent feature patterns across samples within the same cluster. For example, samples in Cluster 16 in Fig. 6.6a exhibit distinct patterns in most feature indices. The lack of alignment among rows indicates weaker intra-cluster similarity. Additionally, the reduced feature values by PCA appear relatively uniform or lack distinctiveness, further suggesting poor feature separability. These observations may explain the low Silhouette and DBCV scores, indicating that each cluster fails to represent a coherent root cause.

For the **quantitative analysis**, we adopt a leave-cluster-out retraining strategy. We perform the analysis on several clusters of the AndroZoo for malware and the Dave for driving data. Following Aghababaeyan *et al.* [16], for each cluster, we retrain the DNN for 5 epochs using 85% of its samples and evaluate the performance gain on the remaining 15%, as well as on all samples from other clusters. If retraining on a specific cluster leads to the most significant improvement on its own remaining samples compared to others, this provides empirical evidence that the cluster represents a unique

Table 6.5 Quantitative fault validation experiment on malware detectors (BasicDNN and DeepDrebin) with AndroZoo dataset and driving models (Dave2V1 and Dave2V2) with Dave dataset. In each row, we retrain the model under test on 85% of the samples in each cluster C_i. We report the accuracy of the retrained model on the remaining 15% of Cluster C_i (the third column) and the accuracy of the retrained model on all other clusters $C_{j \neq i}$ (the fourth column).

Model	Cluster ID C_i	Accuracy on C_i	Accuracy on $C_{j \neq i}$
BasicDNN	Cluster 0	67%	14%
	Cluster 1	73%	16%
	Cluster 2	99%	42%
	Cluster 3	98%	19%
Average		**84%**	**23%**
DeepDrebin	Cluster 0	75%	38%
	Cluster 1	98%	39%
	Cluster 2	71%	42%
	Cluster 3	97%	19%
Average		**85%**	**35%**
Dave2V1	Cluster 0	60%	20%
	Cluster 1	71%	19%
	Cluster 2	83%	23%
	Cluster 3	66%	18%
Average		**70%**	**20%**
Dave2V2	Cluster 0	70%	17%
	Cluster 1	67%	19%
	Cluster 2	95%	20%
	Cluster 3	63%	14%
Average		**74%**	**18%**

and actionable root cause of the faults. We report the results in Table 6.5. All retrained models are significantly more accurate on the clusters for which they were retrained (denoted as C_i), and less accurate on other clusters (denoted as C_j. For example, the average accuracy of BasicDNN on C_i and C_j is 84% and 23%, respectively, where retraining fixes the fault corresponding to cluster C_i but not the fault that C_j represents. Similar observation applies to driving data, where the average accuracy of Dave2V1 on C_i and C_j is 70% and 20%, respectively. Therefore, we conclude that samples in different clusters are mispredicted due to different underlying faults.

Through this combination of visual and empirical evidence, we are confident in claiming that our clustering-based fault identification approach aligns with actual DNN faults.

Finding 1: Validated through both qualitative and quantitative experiments, we conclude that each cluster produced by the best clustering pipeline corresponds to a unique fault and different clusters can represent distinct faults.

6.3.3 RQ1.3 Comparison of different retraining strategies.

Table 6.6 shows the accuracy improvements of four pre-trained models after type I and type II retraining. The detailed retraining process is described in Algorithm 6. The retrained model is evaluated on the hybrid test set, which has the same distribution as the hybrid candidate set from which the test suite is selected. The results are displayed across all budgets and ID ratios, averaged over all metrics and OOD operators. We also include the original accuracy on the hybrid test set before retraining for reference. Among each pair (type I, type II) of retraining test suites, the one with a higher accuracy improvement is highlighted.

Our results reveal distinct trends depending on the domain. For malware data, type I retraining consistently outperforms type II retraining in all 110 cases (55 cases for each dataset). Specifically, type I retraining leads to an average accuracy improvement of 22.37% for AndroZoo and 42.16% for Drebin, compared to 14.75% and 29.87% for type II retraining, respectively. In contrast, the driving datasets exhibit an opposite pattern, where type II outperforms type I in 90 out of 110 cases. For Dave and Udacity, type II retraining achieves improvements of 10.55% and 15.56%, compared to 6.87% and 12.77% under type I. While prior studies, such as Dat [18], have also demonstrated the effectiveness of type II retraining in computer vision tasks, our results reveal that this trend does not generalize uniformly across all data types. These findings highlight that the optimal retraining strategy may be task-dependent.

Table 6.6 Average retraining performance (acc. improvements %) over all metrics and all OOD operators. The first column 'idr' indicates the ID ratio, idr=0% indicates ID+OOD=0%+100%.

	bg=1%		bg=3%		bg=5%		bg=10%		bg=100%		ori
idr	Type I	II	Type I	II	Type I	II	Type I	II	Type I	II	acc.
					AndroZoo						
0%	40.06	28.22	41.96	30.79	45.64	34.90	48.98	36.77	55.77	42.36	42.02
10%	31.68	22.37	37.37	26.55	40.14	30.28	43.58	33.18	50.31	38.15	47.55
20%	25.14	19.99	31.86	22.49	35.68	26.24	37.57	28.52	45.11	33.74	52.69
30%	21.17	16.40	25.06	18.96	29.85	21.49	31.74	23.67	38.64	27.81	58.49
40%	19.42	13.99	21.86	15.85	24.93	17.50	26.89	19.24	33.48	23.45	64.00
50%	16.14	11.06	17.72	12.48	20.22	14.41	22.07	15.28	28.08	19.11	69.61
60%	13.03	8.64	14.11	9.52	15.50	10.97	17.62	11.85	22.76	15.26	75.32
70%	9.15	6.13	10.50	7.11	11.61	8.06	13.26	9.09	17.37	11.33	81.28
80%	5.94	3.47	6.77	4.61	7.47	5.09	8.61	5.57	11.88	7.33	87.12
90%	3.03	1.92	3.54	2.27	4.13	2.61	4.50	2.93	6.40	3.84	92.63
100%	0.27	0.12	0.34	0.14	0.44	0.16	0.56	0.25	0.93	0.29	98.55
Avg	16.93	10.54	19.39	12.38	22.08	14.67	24.21	16.21	29.22	19.94	69.63
					Drebin						
0%	82.49	56.96	82.84	57.49	82.84	60.76	83.99	60.86	85.95	69.13	11.12
10%	72.93	49.57	73.90	52.95	74.14	53.80	75.15	54.30	77.07	61.55	19.82
20%	65.00	46.24	65.68	45.28	66.09	48.35	67.01	48.33	68.67	54.87	28.62
30%	56.45	36.92	58.15	38.66	58.51	41.91	59.44	42.48	61.34	47.79	36.52
40%	47.38	29.49	49.30	32.21	50.70	35.87	51.68	36.40	53.46	40.82	44.60
50%	40.27	25.05	40.72	26.29	41.08	29.20	42.83	30.11	44.59	33.72	53.57
60%	31.90	19.58	32.48	20.27	32.85	23.12	34.20	23.28	35.64	27.16	62.71
70%	23.89	16.03	24.41	15.02	24.71	16.02	25.68	16.58	26.89	20.42	71.45
80%	16.32	10.30	16.89	10.38	17.14	12.13	18.04	12.35	18.94	13.28	79.92
90%	8.71	5.55	9.18	5.35	9.82	5.58	10.01	6.18	10.37	6.78	88.43
100%	1.53	0.78	1.75	0.79	1.89	0.97	2.12	1.15	2.10	0.91	97.02
Avg	40.62	26.90	41.40	27.80	41.97	30.09	42.74	30.38	44.09	34.17	53.98
					Dave						
0%	3.48	13.62	14.83	34.41	27.15	38.64	33.16	44.30	39.15	48.38	8.04
10%	2.31	11.42	14.20	25.57	23.91	29.85	31.85	34.19	26.92	32.50	17.00
20%	0.83	6.90	9.36	19.55	18.55	22.84	21.83	25.93	22.30	28.74	26.01
30%	0.41	5.08	6.27	12.73	12.82	15.51	16.23	17.68	16.51	23.92	35.02
40%	-0.48	3.23	3.02	8.28	6.67	11.18	10.75	12.19	9.65	14.27	43.94
50%	-0.92	1.87	0.42	5.07	2.62	6.00	6.30	6.91	4.09	9.94	52.97
60%	-1.20	0.61	-0.94	3.03	0.05	3.36	2.75	4.77	3.28	2.10	61.92
70%	-1.46	-0.44	-1.41	0.44	-1.11	0.79	-0.28	0.46	3.69	4.81	70.84
80%	-1.69	-1.18	-1.68	-1.01	-1.63	-1.06	-1.55	-0.89	4.47	1.56	79.83
90%	-1.90	-1.81	-1.89	-1.65	-1.89	-1.59	-1.80	-1.46	0.11	-1.38	88.80
100%	0.00	-0.01	-0.00	-0.01	-0.00	-0.02	-0.00	-0.01	-0.01	-0.01	97.83
Avg	-0.06	3.57	3.83	9.67	7.92	11.41	10.84	13.10	11.83	14.98	52.93
					Udacity						
0%	-0.04	21.91	34.93	44.38	44.02	49.16	52.48	57.35	46.77	58.33	9.68
10%	-0.24	17.34	28.89	33.50	38.52	38.92	46.53	46.51	52.25	54.58	18.66
20%	-0.44	14.37	24.35	23.72	28.67	28.46	35.94	36.46	35.54	43.71	27.61
30%	-0.64	12.16	18.87	21.59	23.86	23.19	26.82	27.20	33.75	36.58	36.56
40%	-0.84	7.26	12.43	15.68	17.37	16.44	18.75	21.12	25.20	27.90	45.55
50%	-1.04	4.79	7.81	10.12	10.93	10.80	12.08	13.72	17.73	18.95	54.52
60%	-1.24	2.30	3.78	4.00	5.27	5.10	6.82	7.86	8.72	10.98	63.45
70%	-1.44	0.02	-0.25	0.43	0.36	0.89	1.69	1.67	0.76	1.35	72.40
80%	-1.64	-1.26	-1.51	-1.34	-1.49	-1.15	-1.48	-0.94	0.63	-0.89	81.34
90%	-1.84	-1.90	-1.92	-1.87	-1.90	-1.88	-1.93	-1.89	-0.39	-1.83	90.25
100%	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	99.31
Avg	-0.85	7.00	11.58	13.66	15.06	15.45	17.97	19.01	20.09	22.70	54.48

Moreover, we observe a general trend across both data types: retraining effectiveness improves with increasing test suite budget. For malware datasets, as the budget increases from 1% to 100%, the accuracy improvement of retraining rises from 28.78% to 36.66% for type I, and from 18.72% to 27.06% for type II. Similarly, for driving data, the accuracy improvement of retraining increases from -0.46% to 15.96% for type I and from 5.29% to 18.84% for type II. Since a higher budget allows us to select more samples for retraining, these additional data provide richer feature information that the model may have originally lacked, enhancing its generalizability.

> **Finding 2:** We recommend type I retraining for malware data and type II retraining for driving data. The effectiveness of retraining increases as the selection budget grows.

6.3.4 RQ2. Effectiveness of different metrics in two aspects.

In this research question, we illustrate the results of five studied test selection metrics in the capability of fault detection and retraining guidance.

Capability of fault detection. For each metric, we report the number of clusters obtained with the best clustering pipeline (refer to Section 6.3.1 and Section 6.3.2 for details) across datasets and models, averaging over all four budgets. Recall that each cluster is intended to reflect a common set of features plausible for model mispredictions. More clusters indicate that the selected test suite contains more faults plausible for model misbehaviors.

The results are summarized in Table 6.7. In total, Random, Entropy, DeepGini, Dat, and GD achieve the best performance in fault detection in 11, 31, 33, 10, and 6 out of 88 cases, respectively. Specifically, for malware datasets, the same five selection metrics achieve the highest number of clusters in 3, 14, 22, 5, and 2 out of 44 cases, respectively. For driving datasets, the corresponding results are 8, 16, 11, 5, and 4 out of 44 cases, respectively. In terms of the number of best-performing cases, DeepGini and Entropy achieve the best performance in fault detection for malware and driving data, respectively. In terms of the average number of clusters, for malware data, Entropy achieves the best average values in 3 out of 4 cases, whereas DeepGini achieves the best performance under the 1 left out case (i.e., Drebin dataset with BasicDNN under test). For driving data, although Random and Dat slightly outperform others in Dave datasets in the average number of clusters. They show more inferior performance compared to DeepGini in the Udacity dataset. Specifically, the average number of clusters across

Table 6.7 Number of clusters across datasets and models, averaging over all budgets.

idr	Rand	Ent	Gini	Dat	GD	Rand	Ent	Gini	Dat	GD
	AndroZoo (BasicDNN)					AndroZoo (DeepDrebin)				
0%	13.75	13.25	13.25	12	9.5	12.75	12.75	13	10.25	9.75
10%	9.75	13	11.5	12.25	11	8.75	11.75	11.75	12	8.75
20%	14	11.25	11.25	12.25	11	9.25	10.75	11	10	10
30%	9.5	10.75	10	11.25	9.25	10	10.5	10.75	10	9.5
40%	9.25	12.25	11.75	12	9.5	7.75	10.5	9.5	12	10.5
50%	11.25	12.5	13.5	10	10.25	9.25	9	9.5	11.25	10.25
60%	10.75	9	10.25	7.75	11	8.75	8.5	7.75	10	9.75
70%	10	9.75	9.75	7.25	11.5	7.75	8	8.25	7.5	5.75
80%	8.5	11	10.5	8.75	8.75	7	8	7.75	6.5	6.75
90%	6.5	10.75	10.5	6.5	4.25	6.5	8	7.5	6.75	5.25
100%	1	7.5	8.25	4	0	0.25	2.75	3.25	2.25	0.75
Avg	9.48	11	10.95	9.45	8.73	8	9.14	9.09	8.95	7.91
	Drebin (BasicDNN)					Drebin (DeepDrebin)				
0%	10.25	14.5	11.5	11.75	8.5	13	11.75	10.75	11.25	10
10%	11.25	11.75	13.25	10.25	8.25	10.75	14.5	12.75	9.25	10
20%	9.5	13.25	13.75	10.25	7.75	10.25	12.25	11.5	8.25	9.25
30%	10.75	13.75	13.75	8.75	7.5	7.5	14.75	11.75	7.5	7.25
40%	10.75	11.5	14	9	9.75	7.5	13.75	13	7.75	10.5
50%	9	13	14.5	12.5	7.5	12.25	12	13.25	8.25	8.5
60%	7.5	9.75	12.5	8	9.75	8.5	10.25	11.75	8.5	8.25
70%	8.25	13	15.5	8	7.75	7.25	11.5	12	7.25	6
80%	6	12.5	12.5	8.5	7.75	7	13.75	14	8.25	7
90%	5.5	13.5	14.5	10.5	7.25	7.75	11.75	13.75	6.5	4.75
100%	3.25	8.75	7.25	4.75	2	2	6.75	7.25	3.75	1.75
Avg	8.36	12.3	13	9.3	7.61	8.52	12.09	11.98	7.86	7.57
	Dave (Dave2V1)					Dave (Dave2V2)				
0%	73.75	92.5	69.75	75.75	42	85.25	86.25	83.5	75.75	84.5
10%	87.25	40.25	49	49.25	72.25	73.25	48.25	60.75	73.5	63.25
20%	75	43	54	52.25	79.25	69.75	29	57.5	61.75	69.25
30%	70.25	52.25	46.5	52.5	64	79	37	38	60.75	70.75
40%	60.75	42.5	41.25	42.75	58.25	54	45.25	50.25	48.75	51.25
50%	54.75	55.25	52	43.5	39.5	62.25	53.25	39.5	47.25	44.25
60%	41.75	47.5	36.75	41.25	54.5	50	40.75	49	52.25	49.75
70%	47.5	41	40.75	36.75	36.75	29.5	51.75	47	48.25	40.25
80%	34	34	40	36.25	30.75	33.25	37.25	47.75	45	28.25
90%	20.5	41	39.75	34.5	22	20	36	41.25	40.5	20.25
100%	5	30.25	32.5	16.75	5.25	4.5	33.25	32.25	10.75	5.5
Avg	51.86	47.23	45.66	43.77	45.86	50.98	45.27	49.70	51.32	47.93
	Udacity (Dave2V1)					Udacity (Dave2V2)				
0%	103.5	114.75	104	87.5	105	93.25	95.5	106.25	59.25	102
10%	91.25	76.25	88.25	96	74.25	82.5	93	94.5	68.25	84
20%	85.25	70	81.5	82.5	79.5	80.5	83.5	89.75	83	67.25
30%	72.5	73.25	68.5	65.75	70	67.75	75.5	86.5	58.75	61.75
40%	66.5	73	66	57.5	68.75	79	78.25	79.5	80.5	61.5
50%	70.25	51.25	70.75	51.25	56	60.75	44.25	72.5	57.25	51.75
60%	37	50.75	44.25	52.25	46.25	42.75	47.75	39.5	38	53.25
70%	43.75	45.75	51.5	27.75	33.75	35.25	46.25	38.75	36.75	31.25
80%	28	36.5	32.75	25.75	31.25	26	31.5	35.75	25.75	33.75
90%	14.5	23.75	28	16	21.25	20.75	25.75	23.75	11.75	16.25
100%	1.25	3	4	2.5	1	0.25	3.75	3.5	2.25	1.75
Avg	55.8	56.2	58.14	51.34	53.36	53.52	56.82	60.93	47.41	51.32

both driving datasets and models is 53.61, 53.04, and 48.46 for DeepGini, Random, and Dat, respectively. Therefore, for the capability of fault detection, we recommend DeepGini and Entropy as the preferred selection metrics.

Table 6.8 Accuracy improvements (%) of retraining (type I for malware data, type II for driving data) of the five selection metrics under 11 ID+OOD distributions, averaged over all settings (numbers shown in the parentheses). We highlight the best value in gray (for improvements) and yellow (for declines).

idr	Ran	Ent	Gini	Dat	GD	Ran	Ent	Gini	Dat	GD
	AndroZoo (56)					Drebin (40)				
0%	47.66	48.69	47.30	40.69	47.69	84.62	81.63	84.72	77.17	85.17
10%	43.19	42.57	41.37	33.59	43.35	75.37	73.13	75.93	67.92	76.08
20%	38.28	35.71	33.25	28.24	39.10	67.40	64.99	67.50	60.64	67.32
30%	32.69	28.25	29.75	23.62	32.64	59.80	57.60	58.60	52.58	59.92
40%	27.46	27.10	25.80	20.45	27.51	51.51	49.71	48.37	45.71	51.60
50%	23.44	22.55	21.50	16.65	23.17	42.68	41.40	38.01	39.69	43.01
60%	18.65	17.78	16.92	12.49	18.73	34.12	31.92	30.12	31.78	34.35
70%	13.50	13.26	12.83	8.99	13.53	25.75	23.15	22.45	23.88	25.68
80%	9.16	7.61	8.65	5.78	8.99	17.56	16.27	16.07	15.75	18.11
90%	4.58	4.59	4.71	3.25	4.49	9.43	9.05	9.07	8.80	9.39
100%	0.47	0.84	0.80	0.27	0.46	1.52	1.90	2.35	1.73	1.61
Avg	23.55	22.63	22.08	17.64	23.61	42.71	40.98	41.2	38.69	42.93
	Dave (40)					Udacity (40)				
0%	30.34	37.30	34.98	28.28	32.83	39.05	46.53	46.92	46.97	36.53
10%	21.49	24.32	28.84	24.51	27.12	33.11	32.67	35.00	39.92	29.65
20%	21.93	15.12	17.86	18.61	20.52	26.65	23.61	24.04	29.30	25.15
30%	14.16	10.54	12.99	11.06	14.98	22.06	18.30	18.52	25.49	20.82
40%	10.47	5.84	7.35	8.74	11.19	15.55	14.50	12.88	18.63	14.06
50%	5.36	4.58	4.81	4.27	5.77	10.18	8.96	8.77	11.46	9.93
60%	3.27	2.11	1.77	4.42	3.14	5.48	5.42	4.61	4.62	3.94
70%	0.67	0.20	0.10	-0.07	0.68	1.49	1.03	0.71	0.16	0.36
80%	-1.12	-1.04	-1.00	-0.91	-1.09	-0.94	-0.91	-1.07	-1.46	-1.47
90%	-1.81	-1.60	-1.59	-1.28	-1.84	-1.87	-1.86	-1.88	-1.90	-1.90
100%	-0.01	-0.01	-0.01	-0.02	-0.01	0.01	0.01	0.01	0.01	0.02
Avg	9.52	8.85	9.65	8.87	10.30	13.71	13.48	13.50	15.75	12.46

Finding 3: In terms of the number of best-performing cases, DeepGini and Entropy achieve the best performance in fault detection for malware and driving data, respectively.

Finding 4: In terms of the average number of clusters, Entropy and DeepGini achieve the best performance for malware and driving data, respectively.

Capability of retraining guidance. We report the accuracy improvements in numerical values of the best retraining process (i.e., type I for malware data and type II for driving data) in Table 6.8. The total number of settings is listed in parentheses, for example, there are 40 settings for Drebin (2 models x 5 operators x 4 budgets), and 56 for AndroZoo (2 models x 7 operators x 4 budgets). The values of Table 6.8 are averaged over all settings as listed in parentheses.

Across all datasets, Random, Entropy, DeepGini, Dat, and GD achieve the highest accuracy improvements in 9, 7, 4, 9, and 18 cases, respectively. Specifically, for malware datasets, the counts are 5, 3, 2, 0, and 12, while for driving datasets, they are 4, 4, 2, 9, and 6 cases. For the three datasets, AndroZoo, Drebin, and Dave, GD demonstrates the greatest average improvements, achieving 23.61%, 42.93%, and 10.30%, respectively, whereas Dat achieves the best performance in the Udacity dataset.

Additionally, we observe that retraining provides limited benefits for high ID ratios, as indicated by relatively low values in malware datasets and negative values in driving datasets. This is likely because the pre-trained model has already acquired sufficient ID features. In general, we suggest selecting the test suite with greater diversity since GD gives the best performance in three out of four datasets, covering different data distributions for both RGB images and Android packages.

Finding 5: In general, we suggest using GD for retraining, achieving 33.27% and 11.38% accuracy improvements on average for malware and driving datasets, respectively.
Finding 6: Retraining provides limited benefits for high ID ratios, regardless of data types.

6.3.5 RQ3. Effectiveness of our proposed testing framework.

Our empirical findings recommend DeepGini, Entropy, and GD as candidate selection metrics. Recall that DeepGini and Entropy stand out in the capability of fault detection, whereas GD excels at retraining guidance in most cases. To select a test suite that excels at both criteria, we conduct ensemble selection with DeepGini and Entropy as the input-specific metrics and GD as the suite-wise metric. In addition, we have designed two variants (labeled as V1 and V2) for comparison to validate the effectiveness of each part (i.e., scale unification, ensemble selection) of our proposed testing framework.

This section consists of two parts. The first part presents a comparative analysis between our method and the best-performing baseline for each dataset. While the best baseline may vary depending on the evaluation criterion, the comparison is conducted across both criteria to ensure fairness. The second part focuses on an ablation study, comparing our method with two variants. Statistical analysis has been integrated into both parts.

Table 6.9 Fault detection results of our method, two variants (i.e., V1, V2), and five baselines (i.e., Rand, Ent, Gini, Dat, GD). Darker cells signify superior performance, while various colors denote the statistical significance among eight studied test selection metrics for each ID+OOD ratio (determined by the Scott-Knott ESD test with a p-value≤ 0.05).

idr	Rand	Ent	Gini	Dat	GD	V1	V2	Ours	Rand	Ent	Gini	Dat	GD	V1	V2	Ours
	AndroZoo (56)								Drebin (40)							
0%	13	13	13	11	**10**	13	14	18	**12**	13	**11**	**12**	9	15	15	20
10%	**9**	12	12	12	**10**	13	12	16	11	13	13	**10**	**9**	13	14	18
20%	12	11	11	11	10	13	12	13	10	13	13	**9**	8	15	14	14
30%	10	11	10	11	9	11	11	12	9	14	13	**8**	7	13	14	15
40%	**8**	11	11	12	10	12	11	13	9	13	14	8	**10**	13	13	15
50%	10	11	12	11	10	10	11	12	**11**	12	14	**10**	8	14	12	15
60%	**10**	9	9	9	10	12	11	13	8	**10**	12	8	9	13	12	15
70%	9	9	9	**7**	9	10	10	11	8	12	14	8	7	**10**	12	15
80%	8	10	9	8	8	8	9	10	6	13	13	8	**7**	11	10	13
90%	**6**	9	9	**7**	5	8	9	10	7	13	14	**8**	6	**9**	10	10
100%	1	5	6	3	0	**2**	3	5	**3**	8	7	4	2	**2**	**2**	**2**
Avg	9	10	10	9	8	10	10	12	8	12	12	9	8	11	12	14
	Dave (40)								Udacity (40)							
0%	80	89	77	76	**63**	101	84	109	98	105	105	**73**	104	111	99	125
10%	80	44	**55**	61	68	85	81	98	87	85	91	**82**	**79**	98	**80**	110
20%	72	**36**	56	69	74	72	68	84	83	77	86	83	73	88	75	95
30%	75	**45**	**42**	73	67	64	57	72	70	74	78	74	66	72	**56**	87
40%	57	44	46	56	55	49	44	65	73	76	73	69	65	56	**42**	74
50%	58	54	46	45	42	38	**30**	50	66	48	72	54	54	**41**	36	59
60%	46	44	43	59	52	34	32	44	40	49	42	45	50	37	**26**	47
70%	38	46	44	46	38	**26**	18	32	40	46	45	32	32	**26**	20	35
80%	34	36	44	44	30	**19**	**19**	**19**	27	34	34	26	32	22	**16**	24
90%	20	38	40	38	21	15	15	15	18	25	26	14	19	**11**	9	**10**
100%	5	32	32	14	5	8	**6**	**7**	1	3	4	2	**1**	2	**1**	2
Avg	51	46	48	53	47	46	41	54	55	57	60	50	52	51	42	61

Table 6.10 Accuracy improvements of retraining of our method, two variants (i.e., V1, V2), and five baselines (i.e., Rand, Ent, Gini, Dat, GD). Cell color denotes the statistical significance by the Scott-Knott ESD test.

idr	Rand	Ent	Gini	Dat	GD	V1	V2	Ours
				AndroZoo (56)				
0%	47.66	48.69	47.3	40.69	47.69	48.11	46.53	50.5
10%	43.19	42.57	41.37	33.59	43.35	42.01	40.86	43.48
20%	38.28	35.71	33.25	28.24	39.1	36.94	35.87	39.81
30%	32.69	28.25	29.75	23.62	32.64	31.43	30.68	34.15
40%	27.46	27.1	25.8	20.45	27.51	27.9	24.96	28.23
50%	23.44	22.55	21.5	16.65	23.17	23.63	19.2	25.58
60%	18.65	17.78	16.92	12.49	18.73	15.39	14.69	20.26
70%	13.5	13.26	12.83	8.99	13.53	13.25	11.54	14.54
80%	9.16	7.61	8.65	5.78	8.99	8.55	7.47	9.59
90%	4.58	4.59	4.71	3.25	4.49	3.93	4.0	4.44
100%	0.47	0.84	0.8	0.27	0.46	0.61	0.6	0.63
Avg	23.55	22.63	22.08	17.64	23.61	22.88	21.49	24.65
				Drebin (40)				
0%	84.62	81.63	84.72	77.17	85.17	83.98	78.6	86.21
10%	75.37	73.13	75.93	67.92	76.08	75.23	70.92	77.08
20%	67.4	64.99	67.5	60.64	67.32	66.88	63.23	66.38
30%	59.8	57.6	58.6	52.58	59.92	59.61	56.06	60.84
40%	51.51	49.71	48.37	45.71	51.6	51.06	49.08	52.53
50%	42.68	41.4	38.01	39.69	43.01	39.9	39.68	43.62
60%	34.12	31.92	30.12	31.78	34.35	28.89	26.05	34.8
70%	25.75	23.15	22.45	23.88	25.68	21.26	18.94	26.25
80%	17.56	16.27	16.07	15.75	18.11	15.03	13.68	18.69
90%	9.43	9.05	9.07	8.8	9.39	8.39	7.57	9.0
100%	1.52	1.9	2.35	1.73	1.61	2.12	1.9	2.34
Avg	42.71	40.98	41.2	38.69	42.93	41.12	38.7	43.43
				Dave (40)				
0%	30.34	37.3	34.98	**28.28**	32.83	39.31	**26.51**	40.88
10%	21.49	24.32	28.84	24.51	27.12	30.23	27.33	32.41
20%	21.93	**15.12**	17.86	18.61	20.52	24.11	20.25	25.87
30%	14.16	**10.54**	12.99	**11.07**	14.98	12.82	**11.14**	17.3
40%	10.47	**5.84**	7.35	8.74	11.19	3.72	**5.49**	12.06
50%	5.36	4.58	4.81	4.27	5.78	1.24	1.76	6.05
60%	3.27	2.11	1.77	4.42	3.14	-1.01	**-0.16**	2.66
70%	0.67	0.2	0.1	-0.07	0.67	-1.18	-0.95	-0.02
80%	-1.12	-1.04	-1.0	-0.91	-1.09	-1.67	-1.66	-1.32
90%	-1.81	-1.61	-1.59	-1.28	-1.84	**-1.91**	**-1.9**	-1.65
100%	-0.01	-0.01	-0.01	-0.02	-0.01	-0.0	-0.01	-0.01
Avg	9.52	8.85	9.65	8.87	10.3	9.6	7.98	12.2
				Udacity (40)				
0%	39.05	46.53	46.92	46.97	36.53	48.75	36.82	52.51
10%	33.11	32.67	35.0	39.92	29.65	41.51	36.58	42.85
20%	26.65	23.61	24.04	29.3	25.15	34.06	29.34	33.08
30%	22.06	18.3	18.52	25.49	20.82	25.47	22.71	27.26
40%	15.55	14.5	12.88	18.63	14.06	16.62	14.81	20.49
50%	10.18	8.96	8.77	11.46	9.93	**3.6**	5.81	11.56
60%	5.48	5.42	4.61	4.62	3.94	**1.13**	**1.62**	6.61
70%	1.49	1.03	0.71	0.16	0.36	**-0.84**	-1.23	1.98
80%	-0.94	-0.91	-1.07	-1.46	-1.47	-1.67	-1.7	-1.69
90%	-1.87	-1.86	-1.88	-1.9	-1.9	-1.91	-1.91	-1.9
100%	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
Avg	13.71	13.48	13.5	15.75	12.46	15.16	12.99	17.52

Statistical Significance. In this section, we conduct a statistical analysis to provide an overall ranking of all methods examined in this paper, including the five baselines (i.e., Random, Entropy, DeepGini, Dat, and GD) from RQ2 and the three newly proposed metrics (i.e., V1, V2, and our method) from RQ3. To determine the statistical significance of the observed performance differences among multiple test selection metrics, we employ the parametric Scott-Knott Effect Size Difference (ESD) test, which is based on the assumptions of ANalysis Of VAriance (ANOVA) [185]. The Scott-Knott ESD test is a multiple comparison approach that leverages hierarchical clustering to partition these test selection metrics into distinct groups using the mean values. Different groups exhibit statistically significant differences at the predetermined significance level of 0.05 ($\alpha = 0.05$). The Scott-Knott ESD ensures that the magnitude of the difference between metrics within each group is not statistically significant, and the magnitude of the difference between metrics located in different groups is statistically significant.

In Table 6.9 and Table 6.10, we use the same color for metrics within the same group and different colors for different groups. We utilize three distinct colors to underscore the statistical differences among eight test selection metrics for each ID+OOD ratio in terms of both evaluation criteria. Darker colors represent a superior performance. Specifically, the darkest blue denotes the first group, moderate blue indicates the second group, and the lightest blue represents the third group. These metrics are sometimes categorized into more than three groups. If there are at least five groups, the results of the fourth group will be bolded while maintaining a white cell background.

Comparisons with the best baseline. For an efficient comparison, we pick out the best baseline that achieves the best performance under each dataset for each evaluation criterion. We conduct comparisons between the best baseline and our method under both criteria for fairness.

Table 6.9 demonstrates the results of the capability of fault detection. We report the number of clusters across datasets, averaging over all budgets and models. The process to obtain the number of clusters can be found in Algorithm 5. For this criterion, we identify the best baseline based on the detailed values from Table 6.7 as well as their statistical significance from Table 6.9. Entropy, DeepGini, Dat, and DeepGini provide the best results for AndroZoo, Drebin, Dave, and Udacity datasets, respectively. Moreover, comparing within the five baselines, we note that Entropy, DeepGini, GD, Dat, and Random are located in the best statistical group in 26, 24, 12, 6, and 15 cases, respectively. This validates the finding from RQ2, which suggests employing the Entropy and DeepGini for fault detection. Table 6.10 shows the results

of the capability of retraining guidance. We report the accuracy improvements after the type I retraining for malware data and type II retraining for driving data, averaging over all budgets, models, and OOD operators. For this criterion, the best-performing baselines are GD for the AndroZoo, Drebin, and Dave datasets, and Dat for the Udacity dataset. Among the five baselines, Entropy, DeepGini, GD, Dat, and Random, the number of times each is ranked in the best statistical group is 15, 18, 28, 10, and 33 cases, respectively. Notably, Random appears in the top statistical group more frequently than GD, primarily due to its strong performance on the Udacity dataset. However, when focusing solely on cases with positive accuracy improvements (low ID ratios), GD and Random appear in the best group with equal frequency across the other three datasets, with GD achieving higher average accuracy improvements. This also complements the findings in RQ2 regarding retraining guidance, which suggests using GD for retraining in general.

Capability of fault detection: In this part, we compare our method with the best fault detection baselines along two criteria. Overall, our methodology demonstrates statistically significant improvements over the best baselines (in general, DeepGini and Entropy for fault detection, GD for retraining guidance) in both evaluation criteria. We elaborate details of individual datasets below. In the Udacity dataset, our method detects a larger number of faults (61) than Entropy (57) and DeepGini (60) on average. Ours significantly outperforms both methods in 5 out of 11 cases. For the AndroZoo dataset, compared with DeepGini (10) and Entropy (10), our method (12) detects more faults on average. Ours demonstrates statistically significant superiority in 9 out of 11 cases compared to both. Similar results apply to the Drebin and Dave datasets. In terms of retraining guidance, out of 11 cases, our method significantly outperforms DeepGini in 5, 7, 8, and 8 cases, and outperforms Entropy in 5, 11, 8, and 8 cases, with average additional improvements of 2.57%, 2.23%, 2.55%, and 4.02% (DeepGini), and 2.02%, 2.45%, 3.35%, and 4.04% (Entropy) for AndroZoo, Drebin, Dave, and Udacity dataset, respectively.

In general, the results suggest that our method statistically outperforms the best fault detection baselines in terms of both the number of clusters and the accuracy improvements.

> **Finding 7:** Compared with the best fault detection baselines (DeepGini and Entropy), our method demonstrates a statistically significant improvement in both evaluation criteria.

Capability of retraining guidance: This section provides a comparative analysis between the best baselines in retraining guidance and our method. Compared with GD in the AndroZoo dataset, our method demonstrates statistically significant accuracy improvements in 3 out of 11 cases, with an additional performance gain of 1.04%. Notably, in fault detection, our method significantly outperforms GD in all 11 cases, detecting 4 extra faults on average. For the Drebin dataset, our method significantly surpasses the best baseline GD in 2 out of 11 cases, with an extra average accuracy improvement of 0.5%. In the fault detection, our method achieves significant outperformance in all 11 cases, with 6 extra faults on average. Similar results can be observed in driving data. For the Udacity dataset, our method outperforms the best baseline Dat significantly in 4 out of 11 cases in terms of retraining guidance (with an additional average improvement of 1.77%), and in 5 out of 11 cases in the fault detection (detecting 11 extra faults). For the Dave dataset, our method outperforms the best baseline GD by a margin of 1.9% accuracy. Statistically significant outperformance is observed in 6 out of 11 cases. Similarly, in fault detection, our method outperforms GD in 7 out of 11 cases with detecting 7 extra faults. These results suggest that, in general, our method consistently and significantly outperforms the best retraining baseline. Additionally, the results also highlights that no baseline achieves top performance significantly across both evaluation criteria simultaneously.

> **Finding 8:** Compared with the best retraining baselines, our method demonstrates a statistically significant improvement in both evaluation criteria.

Ablation Study. In this part, we conduct an ablation study with statistical analysis to examine the effectiveness of each part of our proposed methods. The results are summarized in Table 6.9 and Table 6.10. V1 (Gini+GD) represents the first variant of our method, which uses DeepGini only as the input-specific metric and GD as the suite-wise metric. V2 (Gini+Ent+GD) is the second variant, using DeepGini and Entropy as input-specific metrics, with GD as the suite-wise metric. The only difference between our method and V2 is that V2 does not leverage the p-value scale unification, i.e., it directly adds the DeepGini score and Entropy score with equal weights.

The results demonstrate three important observations. The first observation is the importance of scale unification. This can be observed by comparing V2 (Gini+Ent+GD) with our method. For fault detection, we can see that V2 gives poorer performance, detecting an average of 2, 2, 13, and 19 fewer faults than our method

in AndroZoo, Drebin, Dave, and Udacity, respectively. Out of 44 cases, our method shows a statistically significant improvement over V2 in 37 cases, in which 11, 7, 8, and 11 cases are from AndroZoo, Drebin, Dave, and Udacity datasets, respectively. In terms of retraining guidance, our method gives higher accuracy improvements by a margin of 3.16%, 4.73%, 4.22%, and 4.53% in AndroZoo, Drebin, Dave, and Udacity datasets, respectively. In statistical significance, ours outperforms V2 in 36 out of 44 cases, with 7, 11, 10, and 8 cases in each dataset.

Also, the outperformance of our method over V1 (Gini+GD) indicates the usefulness of involving extra beneficial metrics (i.e., Entropy detects the highest number of faults in 3 out of 4 cases for malware data) to the ensemble-based testing framework. Specifically, in fault detection, our method reaches a better performance with statistical significance in a total of 34 out of 44 cases, with 10, 9, 7, and 8 cases in each dataset. The extra average improvement over V1 is approximately 6 faults. In terms of retraining guidance, our method achieves a statistically significant improvement over V1 in a total of 22 out of 44 cases, with 5, 5, 7, and 5 cases in each dataset. The additional average improvement is 1.77%, 1.31%, 2.6%, and 2.36%, respectively.

The third observation highlights the effectiveness of ensemble selection (i.e., considering consensus decisions), as demonstrated by comparing our method with Entropy, DeepGini, and GD across both evaluation criteria. Compared with Entropy and DeepGini, which serve as the best baselines for fault detection, our method demonstrates statistically significant improvements in 26 and 27 cases (out of 44 cases), detecting 4 and 3 additional faults on average, compared to Entropy and DeepGini, respectively. Furthermore, our method achieves a substantial improvement in retaining guidance. Out of 44 cases, there are 31 and 28 statistically significant cases, with extra accuracy improvements of 2.97% and 2.84% on average compared to Entropy and DeepGini, respectively. For GD, which generally serves as the best baseline for retraining guidance, we observe that it does not perform at the top group in fault detection in most cases. Our method outperforms GD in both retraining and fault detection, with an additional accuracy improvement of 2.13% and approximately 7 additional faults on average.

Finding 9: Statistically comparing V2 with our method reveals the importance of scale unification, while our superiority over V1 demonstrates the benefits of incorporating additional useful metrics.
Finding 10: The outperformance of our method over Entropy, DeepGini, and GD indicates the advantage of ensemble-based selection.

6.4 Discussion

6.4.1 Threats to Validity

Internal Validity. The threats to internal validity can be attributed to (1) the implementation quality of studied methods and the reproducibility of our methods. To alleviate these concerns, we carefully follow the publicly available code provided by the original authors. In cases where the authors did not provide implementations (i.e., Entropy), we faithfully followed the method's description to implement it ourselves. Additionally, we open-source our entire implementation to facilitate checking; (2) Effective manipulation of hyperparameters. To alleviate this, for the baselines, we follow the optimal hyperparameter values described in the original papers. For the clustering pipeline, retraining process, and our proposed methodology, we use the recommended selection method with a sufficient number of candidate values to determine the optimal values, as described in the section 6.2.5.

External Validity. External validity refers to the extent to which our experiment conclusions can be generalized [186]. The threats to such validity include the selection of datasets and models. For (1) the selected datasets, we alleviate this by conducting experiments on two different domains, malware detection and autonomous driving. In each domain, we select two widely used and representative datasets. For autonomous driving, we select two real-world driving datasets that are commonly used in both academic research [4, 187, 27] and community challenges [180, 188]. For malware detection research, AndroZoo and Drebin datasets are collected from a massive amount of Android applications with various malware families, and they are commonly used due to their large-scale characteristics and accessibility [179, 85]. Regarding (2) the selected models, we alleviate it by selecting two models for each domain, with each model reaching competitive performance in its respective task (accuracy reported in Table 6.1). In summary, our experimental conclusions have been validated across two distinct domains, utilizing four datasets and models. Therefore, the findings are

expected to be generalized to other real-world driving data and steering classification models, as well as other Android packages and malware detectors.

Construct Validity. Construct validity refers to whether the study correctly identifies the operational measures for the concepts being investigated [186]. In our context, this means the selected evaluation metrics must reliably reflect the intended effects. In this paper, our goal is to design a test selection metric that can effectively select a test suite suitable for various usage scenarios. Our focus lies in two key aspects: (1) detecting the root cause of model faults, and (2) guiding the retraining to enhance model performance [81]. To accurately assess these two aspects, we adopt two distinct evaluation criteria: the number of clusters in feature space for the misprediction set and the accuracy improvements after retraining. These criteria are widely used in the literature [16, 18, 81]. As for the first criterion, we follow the three-step clustering approach proposed by [16, 165], which includes feature extraction, dimensionality reduction, and clustering. For each data type, we empirically identify the most effective clustering pipeline. Our qualitative and quantitative analyses both demonstrate that each cluster produced by the best clustering pipeline can accurately represent a unique fault, and different clusters represent distinct faults (see Sections 6.3.1 and 6.3.2 for details). Hence, the number of clusters is expected to accurately represent the number of root causes of DNN failures. Regarding the second criterion, the accuracy improvement on the corresponding hybrid test set is quite a straightforward measurement of retraining guidance in model enhancement.

6.4.2 Limitations and Future Work

The ensemble framework combines multiple metrics, each capturing different aspects of model behavior. It is without doubt that there will be interaction effects between metrics. To ensure that the selected metrics provide complementary rather than overlapping guidance. In future work, we aim to design an automated process to avoid doing ablation for each metric's selection. Specifically, we can incorporate 'automated correlation analysis' (e.g., Spearman correlation [189] of metrics' scores) into the testing framework, and automatically drop redundant metrics.

6.5 Chapter Summary

In this chapter, we examine the effectiveness and reliability of current testing approaches on autonomous driving and malware detection domains. We identify three

problems existing in the current literature: a lack of the consideration of consensus decision (i.e., selecting inputs based on a single metric, which only captures a single aspect of the test input), insufficient evaluation criteria, and limited evaluation tasks and datasets. Our proposed method addresses them correspondingly by (1) performing ensemble-based testing through granularity and scale unification, (2) examining the effectiveness of test selection metrics with two criteria, assessing both the capability of fault detection and the capability of retraining guidance, and (3) evaluating on two different domains, autonomous driving and malware detection. Our experiment results and statistical analysis demonstrate the effectiveness of our method compared with five baselines along both evaluation criteria.

Chapter 7

Conclusion and Future Work

This thesis introduces innovative automated testing approaches for intelligent software systems, aiming at overcoming challenges in deep learning testing, containing four specific topics: adversarial test input generation in autonomous driving, transferable test input generation in autonomous driving, valid test input generation, and ensemble-based test input selection in multiple domains. The chapter wraps up with a summary of the contributions made and offers insightful recommendations for future research directions.

7.1 Thesis Contributions

In the area of adversarial test input generation for autonomous driving systems (ADSs), we introduce UniAda, a novel test input generation approach that generates an image-agnostic adversarial perturbation effective for inducing misbehavior on multiple control maneuvers (steering and speed) of the ADS under test (i.e., E2E ADSs). UniAda capitalizes on an intricately designed multi-objective optimization function with the Adaptive Weighting Scheme (AWS), enabling the concurrent optimization of diverse objectives. Specifically, AWS is capable of balancing varying objective weights dynamically during the perturbation searching process, which saves the effort of exhaustive grid searches for objective weights. Meanwhile, UniAda applies white-box iterative gradient descent to optimize the joint objective. Validated with both simulated and real-world driving data, UniAda outperforms five benchmarks across two metrics, inducing steering and speed deviations from 3.54° to 29° and 11 km/h to 22 km/h on average. This systematic approach establishes UniAda as a proven technique for adversarial attacks on modern DL-based E2E ADSs.

In the area of transferable test input generation in autonomous driving, we conduct an empirical study that investigates the effectiveness of different transferability-enhancement techniques proposed in other fields (e.g., simple image classification). Our empirical findings provide insightful recommendations for researchers who wish to utilize these techniques in ADSs. We identify nine enhancements in total: three input transformations and six attack objectives, which are applicable to E2E steering models. We evaluate their effectiveness in both white-box and black-box attacks using two datasets and four steering models. For black-box and timely attacks, *Resizing+Translation* is recommended. For high effectiveness in both attack types, *Pred+Attn* is recommended. However, we do not recommend *Feat+Attn* for either attack type. Additionally, our runtime efficiency assessment also demonstrates that Resizing+Translation takes only half the runtime of Pred+Attn, making it more practical for real-world black-box attacks.

In the area of valid test input generation, we conduct an empirical study and then, based on our empirical findings, we propose a novel testing framework that can generate more valid test inputs. For the empirical study, we comprehensively assess four automated Input Validators (IVs) and six test selection metrics. Our findings show that the accuracy of IVs ranges from 49% to 77%. Distance-based IVs generally outperform reconstruction-based and density-based IVs for both classification and regression tasks. Additionally, we analyze the test selection metrics' score achieved by valid and invalid inputs separately. The results reveal that invalid inputs consistently inflate uncertainty-based metrics. Based on the findings, we enhance existing testing frameworks by incorporating distribution awareness through joint optimization. The results demonstrate our framework leads to a 2% to 10% increase in the number of valid inputs, which establishes our method as an effective technique for valid test input generation.

Although existing test input generation techniques have demonstrated superior results, we still observe three key limitations: 1. Lack of consensus decision-making across different metrics; 2. Insufficient effectiveness evaluation; 3. Limited scope of tasks and datasets. To address these limitations, we further propose a novel testing framework by performing ensemble-based testing through granularity and scale unification, and examining the effectiveness of test selection metrics on both the capability of cause detection and the capability of retraining guidance. Furthermore, we conduct our experiments on two different domains. Validated on two Android malware and two real-world driving datasets, our observations reveal that none of the five studied metrics achieve statistically significant top performance across both

evaluation criteria simultaneously. In contrast, our method successfully achieves this. Compared to the best baseline in cause detection, our method identifies at least the same number of causes, while delivering substantial and statistically significant improvements in retraining guidance. The same holds true for the best baseline of retraining guidance. Additionally, our ablation study statistically validates the effectiveness of each component of our method, including scale unification and ensemble-based selection.

7.2 Contributions to Software Engineering

These contributions collectively enhance software engineering practices by providing practical, validated tools and insights that address common challenges.

UniAda testing approach for enhancing the reliability and robustness of ADSs: The proposed UniAda approach aims to identify potential misbehavior of ADSs by generating adversarial test inputs. The generated adversarial samples are expected to reveal misbehavior-inducing features that are able to consistently trigger errors on both speed and steering control maneuvers. By collecting effective adversarial inputs, the robustness of ADSs under test can be further enhanced through adversarial training, thus improving the reliability of the system when deployed in the real world.

Transferable test generation for ADSs allows scalable testing, which enables testing multiple ADSs without redesigning tests for each, efficiently reducing experts' efforts. Our empirical study systematically evaluates techniques from other fields and adapts them to complex systems (i.e., ADSs). This contributes to a deeper understanding of how test generalizability can be improved. These insights inform best practices for testing in environments where model internals may be unknown or partially accessible, thus supporting more flexible and reusable test strategies in autonomous driving.

Meanwhile, promoting test input validity is essential for effective testing. Feeding invalid inputs to the SUT is meaningless, as invalid inputs cannot correctly test the generalized behavior of the SUT. Our proposed testing framework enhances the generation of valid inputs through incorporating distribution awareness. This contribution helps address long-standing concerns in software engineering about the quality and relevance of test data, particularly for machine learning-enabled systems.

Advancing Test Selection through Ensemble Techniques: The fourth contribution tackles the issue of ineffective test selection and insufficient test evaluation. The proposed ensemble-based testing approach are valuable to enhance software quality assurance across various DL-based software, including malware detectors, image

classifiers, and ADSs. The framework also provides more comprehensive evaluation criteria that allow efficient and unbiased assessment across different testing approaches in software engineering.

Ethical considerations. The development of transferable adversarial attacks raises important ethical questions, particularly in the context of safety-critical systems like autonomous driving. While our study is intended to strengthen the testing and robustness evaluation of intelligent systems, such techniques can be misused to craft intentional failures. We emphasize that this research is conducted in the spirit of responsible disclosure and contributes toward building more secure, trustworthy, and testable intelligent systems. All experimental protocols are executed in simulation or offline settings, without interfering with operational and commercialized systems. Practitioners deploying such techniques in real-world engineering pipelines should carefully control access, ensure responsible usage, and conduct thorough risk assessments, especially in systems that interact with the physical world.

7.3 Limitations and Future Research Work

While our approaches for deep learning testing have yielded notable successes, there remain avenues for further enhancement based on our findings.

In the area of adversarial test input generation for ADSs, our current methodology involves the offline generation of a universal, multi-objective adversarial perturbation that effectively induces misbehavior across multiple vehicle control dimensions, such as steering and speed. This approach, embodied in our UniAda framework, leverages multi-objective optimization to craft perturbations that are robust across diverse inputs and driving scenarios. However, our method is designed for vision-based E2E steering models, using RGB images as input and continuous control outputs (e.g., steering angle). Therefore, it may not apply directly to (1) ADSs using non-vision sensors (e.g., LiDAR, radar); (2) modular ADSs with discrete intermediate stages (e.g., perception, planning, and control); (3) closed-source black-box systems with limited access. For future work, we plan to extend UniAda by integrating it into an online testing framework, with the goal of generating adversarial test inputs in real time within a closed-loop environment. This advancement would allow for the timely detection of control failures as the ADS interacts with dynamic driving contexts. To realize this, we envision integrating UniAda with a driving simulator platform (e.g., CARLA, LGSVL) and connecting it directly to an ADS under test. This integration would support continuous adversarial input generation, real-time feedback monitoring, and

adaptive perturbation updates based on system responses, thereby bridging the gap between static offline testing and dynamic, interactive evaluation of safety-critical ADSs.

In the area of transferable test case generation, we present an effectiveness analysis of nine transferability-enhancement methods on E2E ADSs, covering two types: input transformation enhancements and attack objective enhancements. Our study provides valuable findings and serves as a robust benchmark for further advancements in the reliability and robustness of ADSs. Our focus in future work will be to validate these techniques' effectiveness in the real physical world. Experimentation on digital images may overlook several physical constraints, including perturbation imperceptibility, spatial constraints, environmental conditions, fabrication errors, and context sensitivity. To generate physical adversarial examples, future work must incorporate these real-world constraints into the generation algorithm, ensuring that perturbations remain effective, physically realizable, and inconspicuous when applied to tangible objects (e.g., printed signs, adversarial patches). Additionally, it is necessary to conduct physical tests using real vehicles. Such efforts are crucial for transitioning from theoretical attacks to practical tools for evaluating the safety and robustness of intelligent systems under real operational conditions.

In the area of valid test case generation, we present a joint optimization testing framework, which incorporates the in-distribution awareness objective into the testing objective, allowing for generating more semantically valid test inputs that are within the input distribution on which the model was trained. Our experimentation was conducted on image data, which renders limited generalization of our results to other fields. In future work, we aim to extend our approach to other fields to enhance the completeness and understand the generalizability of our findings, such as malware detection using Android packages [92] and sentiment analysis using text data [3].

In the area of test selection and evaluation, our empirical findings provide insights into the effectiveness of existing test selection metrics along two aspects. Based on the findings, we propose a novel ensemble-based testing framework by incorporating effective metrics to make a consensus decision with fused scores. Our method is statistically effective in both fault detection and performance improvement on the SUT across autonomous driving and malware detection domains. However, the current framework relies on a static fusion strategy, where metric weights are determined through exhaustive grid search. As a direction for future work, we aim to explore adaptive weighting mechanisms that dynamically adjust the contribution of each metric based on real-time model feedback (e.g., prediction uncertainty, error patterns). Such

dynamic ensemble strategies may further enhance the generalizability, robustness, and efficiency of test input selection in evolving deployment scenarios.

References

[1] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to End Learning for Self-Driving Cars," *arXiv preprint arXiv:1604.07316*, 2016.

[2] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket." in *Ndss*, vol. 14, 2014, pp. 23–26.

[3] A. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, 2011, pp. 142–150.

[4] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 1–18.

[5] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu *et al.*, "Deepgauge: Multi-granularity testing criteria for deep learning systems," in *Proceedings of the 33rd ACM/IEEE international conference on automated software engineering*, 2018, pp. 120–131.

[6] V. Riccio and P. Tonella, "When and Why Test Generators for Deep Learning Produce Invalid Inputs: An Empirical Study," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 1161–1173.

[7] J. Guo, Y. Jiang, Y. Zhao, Q. Chen, and J. Sun, "Dlfuzz: Differential fuzzing testing of deep learning systems," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 739–743.

[8] S. Kang, R. Feldt, and S. Yoo, "Sinvad: Search-based image space navigation for dnn image classifier test input generation," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, 2020, pp. 521–528.

[9] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *Artificial intelligence safety and security*. Chapman and Hall/CRC, 2018, pp. 99–112.

References

[10] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[11] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "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*, 2018, pp. 132–142.

[12] J. G. Moreno-Torres, T. Raeder, R. Alaiz-Rodríguez, N. V. Chawla, and F. Herrera, "A unifying view on dataset shift in classification," *Pattern recognition*, vol. 45, no. 1, pp. 521–530, 2012.

[13] Y. Feng, Q. Shi, X. Gao, J. Wan, C. Fang, and Z. Chen, "Deepgini: prioritizing massive tests to enhance the robustness of deep neural networks," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2020, pp. 177–188.

[14] D. Wang and Y. Shang, "A new active labeling method for deep learning," in *2014 International joint conference on neural networks (IJCNN)*. IEEE, 2014, pp. 112–119.

[15] J. Kim, R. Feldt, and S. Yoo, "Guiding Deep Learning System Testing using Surprise Adequacy," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 1039–1049.

[16] Z. Aghababaeyan, M. Abdellatif, L. Briand, S. Ramesh, and M. Bagherzadeh, "Black-box testing of deep neural networks through test case diversity," *IEEE Transactions on Software Engineering*, vol. 49, no. 5, pp. 3182–3204, 2023.

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[18] Q. Hu, Y. Guo, M. Cordy, X. Xie, L. Ma, M. Papadakis, and Y. Le Traon, "An empirical study on data distribution-aware test selection for deep learning enhancement," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 4, pp. 1–30, 2022.

[19] M. HUTSON, "Watch just a few self-driving cars stop traffic jams," https://www.science.org/content/article/watch-just-few-self-driving-cars-stop-traffic-jams, 2018.

[20] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE access*, vol. 8, pp. 58 443–58 469, 2020.

[21] A. Tampuu, T. Matiisen, M. Semikin, D. Fishman, and N. Muhammad, "A Survey of End-to-End Driving: Architectures and Training Methods," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 4, pp. 1364–1384, 2020.

[22] E. D. Dickmanns, "The Development of Machine Vision for Road Vehicles in the Last Decade," in *Intelligent Vehicle Symposium, 2002. IEEE*, vol. 1. IEEE, 2002, pp. 268–281.

[23] J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman *et al.*, "A Perception-Driven Autonomous Urban Vehicle," *Journal of Field Robotics*, vol. 25, no. 10, pp. 727–774, 2008.

[24] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-end Driving via Conditional Imitation Learning," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 4693–4700.

[25] F. Codevilla, E. Santana, A. M. López, and A. Gaidon, "Exploring the Limitations of Behavior Cloning for Autonomous Driving," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 9329–9338.

[26] U. dataset, "Using Deep Learning to Predict Steering Angles." https://medium.com/udacity/challenge-2-using-deep-learning-to-predict-steering-angles-f42004a36ff3, 2016.

[27] H. Zhou, W. Li, Z. Kong, J. Guo, Y. Zhang, B. Yu, L. Zhang, and C. Liu, "Deepbillboard: Systematic physical-world testing of autonomous driving systems," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 347–358.

[28] Z. Kong, J. Guo, A. Li, and C. Liu, "Physgan: Generating physical-world-resilient adversarial examples for autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 14 254–14 263.

[29] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing Properties of Neural Networks," *arXiv preprint arXiv:1312.6199*, 2013.

[30] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," *arXiv preprint arXiv:1412.6572*, 2014.

[31] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial Machine Learning at Scale," *arXiv preprint arXiv:1611.01236*, 2016.

[32] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial Examples in the Physical World," in *Artificial intelligence safety and security*. Chapman and Hall/CRC, 2018, pp. 99–112.

[33] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble Adversarial Training: Attacks and Defenses," *arXiv preprint arXiv:1705.07204*, 2017.

[34] Z. Li, M. Pan, T. Zhang, and X. Li, "Testing DNN-based Autonomous Driving Systems under Critical Environmental Conditions," in *International Conference on Machine Learning*. PMLR, 2021, pp. 6471–6482.

[35] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th international conference on software engineering*, 2018, pp. 303–314.

[36] M. von Stein, D. Shriver, and S. Elbaum, "DeepManeuver: Adversarial Test Generation for Trajectory Manipulation of Autonomous Vehicles," *IEEE Transactions on Software Engineering*, 2023.

[37] A. Stocco, M. Weiss, M. Calzana, and P. Tonella, "Misbehaviour Prediction for Autonomous Driving Systems," in *Proceedings of the ACM/IEEE 42nd international conference on software engineering*, 2020, pp. 359–371.

[38] C. Zhang, P. Benz, C. Lin, A. Karjauv, J. Wu, and I. S. Kweon, "A Survey on Universal Adversarial Attack," *arXiv preprint arXiv:2103.01498*, 2021.

[39] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal Adversarial Perturbations," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1765–1773.

[40] Z. Chen, V. Badrinarayanan, C.-Y. Lee, and A. Rabinovich, "GradNorm: Gradient Normalization for Adaptive Loss Balancing in Deep Multitask Networks," in *International Conference on Machine Learning*. PMLR, 2018, pp. 794–803.

[41] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets Robotics: The KITTI Dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[42] Sully-Chen, "Autopilot-Tensorflow," 2016, *https://github.com/SullyChen/Autopilot-TensorFlow*.

[43] J. Zhang, Y. Lou, J. Wang, K. Wu, K. Lu, and X. Jia, "Evaluating Adversarial Attacks on Driving Safety in Vision-Based Autonomous Vehicles," *IEEE Internet of Things Journal*, vol. 9, no. 5, pp. 3443–3456, 2021.

[44] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, 2017, pp. 506–519.

[45] F. Croce and M. Hein, "Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks," in *International conference on machine learning*. PMLR, 2020, pp. 2206–2216.

[46] J. Zhang, J. W. Keung, Y. Xiao, Y. Liao, Y. Li, and X. Ma, "Uniada: Universal adaptive multiobjective adversarial attack for end-to-end autonomous driving systems," *IEEE Transactions on Reliability*, 2024.

[47] C. Xie, Z. Zhang, Y. Zhou, S. Bai, J. Wang, Z. Ren, and A. L. Yuille, "Improving Transferability of Adversarial Examples with Input Diversity," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 2730–2739.

[48] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," in *International Conference on Learning Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=Sys6GJqxl

[49] Z. Jin, J. Zhang, Z. Zhu, and H. Chen, "Benchmarking transferable adversarial attacks." *CoRR*, 2024.

[50] Y. Deng, X. Zheng, T. Zhang, C. Chen, G. Lou, and M. Kim, "An Analysis of Adversarial Attacks and Defenses on Autonomous Driving Models," in *2020 IEEE international conference on pervasive computing and communications (PerCom)*. IEEE, 2020, pp. 1–10.

[51] Y. Dong, T. Pang, H. Su, and J. Zhu, "Evading defenses to transferable adversarial examples by translation-invariant attacks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 4312–4321.

[52] S. Chen, Z. He, C. Sun, J. Yang, and X. Huang, "Universal adversarial attack on attention and the resulting dataset damagenet," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 4, pp. 2188–2197, 2020.

[53] Y. Lu, H. Ren, W. Chai, S. Velipasalar, and Y. Li, "Time-aware and task-transferable adversarial attack for perception of autonomous vehicles," *Pattern Recognition Letters*, vol. 178, pp. 145–152, 2024.

[54] A. S. Hashemi, A. Bär, S. Mozaffari, and T. Fingscheidt, "Improving transferability of generated universal adversarial perturbations for image classification and segmentation," in *Deep Neural Networks and Data for Automated Driving: Robustness, Uncertainty Quantification, and Insights Towards Safety*. Springer International Publishing Cham, 2022, pp. 171–196.

[55] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" *Advances in neural information processing systems*, vol. 27, 2014.

[56] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[57] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 3642–3649.

[58] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[59] D. Ciresan, A. Giusti, L. Gambardella, and J. Schmidhuber, "Deep neural networks segment neuronal membranes in electron microscopy images," *Advances in neural information processing systems*, vol. 25, 2012.

[60] Y. Sun, X. Huang, D. Kroening, J. Sharp, M. Hill, and R. Ashmore, "Deepconcolic: Testing and debugging deep neural networks," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2019, pp. 111–114.

[61] J. H. Hayes and J. Offutt, "Input validation analysis and testing," *Empirical Software Engineering*, vol. 11, pp. 493–522, 2006.

[62] H. Liu and H. B. K. Tan, "Covering code behavior on input validation in functional testing," *Information and Software Technology*, vol. 51, no. 2, pp. 546–553, 2009.

[63] S. Hanna and M. Munro, "Test case generation for semantic-based user input validation of web applications," *International Journal of Web Engineering and Technology*, vol. 13, no. 3, pp. 225–254, 2018.

[64] S. Dola, M. B. Dwyer, and M. L. Soffa, "Distribution-aware testing of neural networks using generative models," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 226–237.

[65] A. Stocco, M. Weiss, M. Calzana, and P. Tonella, "Misbehaviour prediction for autonomous driving systems," in *Proceedings of the ACM/IEEE 42nd international conference on software engineering*, 2020, pp. 359–371.

[66] J. Yang, R. Xu, Z. Qi, and Y. Shi, "Visual anomaly detection for images: A survey," *arXiv preprint arXiv:2109.13157*, 2021.

[67] J. Yang, K. Zhou, Y. Li, and Z. Liu, "Generalized out-of-distribution detection: A survey. arxiv," *arXiv preprint arXiv:2110.11334*, 2021.

[68] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma, "Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications," *arXiv preprint arXiv:1701.05517*, 2017.

[69] W. Hu, J. Gao, B. Li, O. Wu, J. Du, and S. Maybank, "Anomaly detection using local kernel density estimation and context-based regression," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 2, pp. 218–233, 2018.

[70] R. Hinami, T. Mei, and S. Satoh, "Joint detection and recounting of abnormal events by learning deep generic knowledge," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 3619–3627.

[71] L. Zhang, J. Lin, and R. Karim, "Adaptive kernel density-based anomaly detection for nonlinear systems," *Knowledge-Based Systems*, vol. 139, pp. 50–63, 2018.

[72] J. An and S. Cho, "Variational autoencoder based anomaly detection using reconstruction probability," *Special lecture on IE*, vol. 2, no. 1, pp. 1–18, 2015.

[73] X. Xia, X. Pan, N. Li, X. He, L. Ma, X. Zhang, and N. Ding, "Gan-based anomaly detection: A review," *Neurocomputing*, vol. 493, pp. 497–535, 2022.

[74] H. Zenati, C. S. Foo, B. Lecouat, G. Manek, and V. R. Chandrasekhar, "Efficient gan-based anomaly detection," *arXiv preprint arXiv:1802.06222*, 2018.

[75] S. Akcay, A. Atapour-Abarghouei, and T. P. Breckon, "Ganomaly: Semi-supervised anomaly detection via adversarial training," in *Computer Vision–ACCV 2018: 14th Asian Conference on Computer Vision, Perth, Australia, December 2–6, 2018, Revised Selected Papers, Part III 14*. Springer, 2019, pp. 622–637.

[76] W. Liu, W. Luo, D. Lian, and S. Gao, "Future frame prediction for anomaly detection–a new baseline," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6536–6545.

[77] Y. Sun, Y. Ming, X. Zhu, and Y. Li, "Out-of-distribution detection with deep nearest neighbors," in *International Conference on Machine Learning*. PMLR, 2022, pp. 20 827–20 840.

[78] T. Dietterich, "A study of distance-based machine learning algorithms," Ph.D. dissertation, Ph. D. Thesis, computer Science Dept., Oregon State University, 1995.

[79] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft, "Deep one-class classification," in *International conference on machine learning*. PMLR, 2018, pp. 4393–4402.

[80] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Transactions on Software Engineering*, 2020.

[81] M. Weiss and P. Tonella, "Simple techniques work surprisingly well for neural network test prioritization and active learning (replicability study)," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2022, pp. 139–150.

[82] Weiss, Michael and Tonella, Paolo, "Fail-safe execution of deep learning based systems through uncertainty monitoring," in *2021 14th IEEE conference on software testing, verification and validation (ICST)*. IEEE, 2021, pp. 24–35.

[83] D. W. Tory Griggs, "How a Self-Driving Uber Killed a Pedestrian in Arizona," https://www.nytimes.com/interactive/2018/03/20/us/self-driving-uber-pedestrian-killed.html, 2018.

[84] Y. Huang, J. Song, Q. Hu, F. Juefei-Xu, and L. Ma, "Active testing of large language model via multi-stage sampling," 2024. [Online]. Available: https://arxiv.org/abs/2408.03573

[85] D. Li, T. Qiu, S. Chen, Q. Li, and S. Xu, "Can we leverage predictive uncertainty to detect dataset shift and adversarial examples in android malware detection?" in *Proceedings of the 37th Annual Computer Security Applications Conference*, 2021, pp. 596–608.

[86] W. Shen, Y. Li, L. Chen, Y. Han, Y. Zhou, and B. Xu, "Multiple-boundary clustering and prioritization to promote neural network retraining," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, 2020, pp. 410–422.

[87] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software testing, verification and reliability*, vol. 22, no. 2, pp. 67–120, 2012.

[88] H. Fahmy, F. Pastore, M. Bagherzadeh, and L. Briand, "Supporting deep neural network safety analysis and retraining through heatmap-based unsupervised learning," *IEEE Transactions on Reliability*, vol. 70, no. 4, pp. 1641–1657, 2021.

[89] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[90] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, A. Y. Ng *et al.*, "Reading digits in natural images with unsupervised feature learning," in *NIPS workshop on deep learning and unsupervised feature learning*, vol. 2011, no. 2. Granada, 2011, p. 4.

[91] A. Krizhevsky, V. Nair, and G. Hinton, "The cifar-10 dataset." *online: http:// www.cs.toronto.edu/ kriz/ cifar.html*, 2014.

[92] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Androzoo: Collecting millions of android apps for the research community," in *Proceedings of the 13th international conference on mining software repositories*, 2016, pp. 468–471.

[93] S. Kuutti, R. Bowden, Y. Jin, P. Barber, and S. Fallah, "A Survey of Deep Learning Applications to Autonomous Vehicle Control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 2, pp. 712–733, 2020.

[94] K. Chitta, A. Prakash, B. Jaeger, Z. Yu, K. Renz, and A. Geiger, "Transfuser: Imitation with Transformer-based Sensor Fusion for Autonomous Driving," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[95] S. Casas, A. Sadat, and R. Urtasun, "MP3: A Unified Model to Map, Perceive, Predict and Plan," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 14 403–14 412.

[96] D. A. Pomerleau, "ALVINN: An Autonomous Land Vehicle in a Neural Network," *Advances in neural information processing systems*, vol. 1, 1988.

[97] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial Examples: Attacks and Defenses for Deep Learning," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 9, pp. 2805–2824, 2019.

[98] F. Codevilla, A. M. Lopez, V. Koltun, and A. Dosovitskiy, "On Offline Evaluation of Vision-based Driving Models," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 236–251.

[99] F. U. Haq, D. Shin, S. Nejati, and L. C. Briand, "Comparing Offline and Online Testing of Deep Neural Networks: An Autonomous Car Case Study," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 2020, pp. 85–95.

[100] S. Nidhra and J. Dondeti, "Black Box and White Box Testing Techniques-A Literature Review," *International Journal of Embedded Systems and Applications (IJESA)*, vol. 2, no. 2, pp. 29–50, 2012.

[101] H. Wu, S. Yunas, S. Rowlands, W. Ruan, and J. Wahlström, "Adversarial driving: Attacking end-to-end autonomous driving," in *2023 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2023, pp. 1–7.

[102] N. Narodytska and S. P. Kasiviswanathan, "Simple black-box adversarial perturbations for deep networks," *arXiv preprint arXiv:1612.06299*, 2016.

[103] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *2nd International Conference on Learning Representations, ICLR 2014*, 2014.

[104] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III 13*. Springer, 2013, pp. 387–402.

[105] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, and J. Li, "Boosting adversarial attacks with momentum," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 9185–9193.

[106] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into Transferable Adversarial Examples and Black-box Attacks," *arXiv preprint arXiv:1611.02770*, 2016.

[107] F. Tramèr, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "The space of transferable adversarial examples," *arXiv preprint arXiv:1704.03453*, 2017.

[108] N. Papernot, P. McDaniel, and I. Goodfellow, "Transferability in machine learning: From phenomena to black-box attacks using adversarial samples. arxiv 2016," *arXiv preprint arXiv:1605.07277*, 2016.

[109] G. Lin, Z. Pan, X. Zhou, Y. Duan, W. Bai, D. Zhan, L. Zhu, G. Zhao, and T. Li, "Boosting adversarial transferability with shallow-feature attack on sar images," *Remote Sensing*, vol. 15, no. 10, p. 2699, 2023.

[110] D. Wang, W. Yao, T. Jiang, and X. Chen, "Improving transferability of universal adversarial perturbation with feature disruption," *IEEE Transactions on Image Processing*, 2023.

[111] D. Lang, D. Chen, R. Shi, and Y. He, "Attention-guided digital adversarial patches on visual detection," *Security and Communication Networks*, vol. 2021, no. 1, p. 6637936, 2021.

[112] A. M. Staff, J. Zhang, J. Li, J. Xie, E. A. Traiger, J. A. Glomsrud, and K. B. Karolius, "An empirical study on cross-data transferability of adversarial attacks on object detectors." in *AI-Cybersec@ SGAI*, 2021, pp. 38–52.

[113] L. Ma, F. Juefei-Xu, M. Xue, B. Li, L. Li, Y. Liu, and J. Zhao, "Deepct: Tomographic combinatorial testing for deep learning systems," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2019, pp. 614–618.

[114] X. Xie, L. Ma, F. Juefei-Xu, M. Xue, H. Chen, Y. Liu, J. Zhao, B. Li, J. Yin, and S. See, "Deephunter: a coverage-guided fuzz testing framework for deep neural networks," in *Proceedings of the 28th ACM SIGSOFT international symposium on software testing and analysis*, 2019, pp. 146–157.

[115] I. Dunn, H. Pouget, D. Kroening, and T. Melham, "Exposing previously undetectable faults in deep neural networks," in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2021, pp. 56–66.

[116] I. Dunn, L. Hanu, H. Pouget, D. Kroening, and T. Melham, "Evaluating robustness to context-sensitive feature perturbations of different granularities," *arXiv preprint arXiv:2001.11055*, 2020.

[117] V. Riccio and P. Tonella, "Model-based exploration of the frontier of behaviours for deep learning system testing," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 876–888.

[118] J. Kauffmann, L. Ruff, G. Montavon, and K.-R. Müller, "The clever hans effect in anomaly detection," *arXiv preprint arXiv:2006.10609*, 2020.

[119] A. J. Izenman, "Review papers: Recent developments in nonparametric density estimation," *Journal of the american statistical association*, vol. 86, no. 413, pp. 205–224, 1991.

[120] K. Lee, K. Lee, H. Lee, and J. Shin, "A simple unified framework for detecting out-of-distribution samples and adversarial attacks," *Advances in neural information processing systems*, vol. 31, 2018.

[121] V. Sehwag, M. Chiang, and P. Mittal, "Ssd: A unified framework for self-supervised outlier detection," *arXiv preprint arXiv:2103.12051*, 2021.

[122] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 eighth ieee international conference on data mining*. IEEE, 2008, pp. 413–422.

[123] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural computation*, vol. 13, no. 7, pp. 1443–1471, 2001.

[124] T. Pevnỳ, "Loda: Lightweight on-line detector of anomalies," *Machine Learning*, vol. 102, pp. 275–304, 2016.

[125] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 2000, pp. 93–104.

[126] M.-L. Shyu, S.-C. Chen, K. Sarinnapakorn, and L. Chang, "A novel anomaly detection scheme based on principal component classifier," in *Proceedings of the IEEE foundations and new directions of data mining workshop*. IEEE Press, 2003, pp. 172–179.

[127] W. Ma, M. Papadakis, A. Tsakmalis, M. Cordy, and Y. L. Traon, "Test selection for deep learning systems," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 2, pp. 1–22, 2021.

[128] F. Harel-Canada, L. Wang, M. A. Gulzar, Q. Gu, and M. Kim, "Is neuron coverage a meaningful measure for testing deep neural networks?" in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 851–862.

[129] Y. Ovadia, E. Fertig, J. Ren, Z. Nado, D. Sculley, S. Nowozin, J. Dillon, B. Lakshminarayanan, and J. Snoek, "Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift," *Advances in neural information processing systems*, vol. 32, 2019.

[130] X. Zhang, X. Xie, L. Ma, X. Du, Q. Hu, Y. Liu, J. Zhao, and M. Sun, "Towards characterizing adversarial defects of deep learning software from the lens of uncertainty," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 739–751.

[131] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*. PMLR, 2016, pp. 1050–1059.

[132] A. Kendall and Y. Gal, "What uncertainties do we need in bayesian deep learning for computer vision?" *Advances in neural information processing systems*, vol. 30, 2017.

[133] M. A. Ganaie, M. Hu, A. K. Malik, M. Tanveer, and P. N. Suganthan, "Ensemble deep learning: A review," *Engineering Applications of Artificial Intelligence*, vol. 115, p. 105151, 2022.

[134] C. Ju, A. Bibaut, and M. van der Laan, "The relative performance of ensemble methods with deep convolutional neural networks for image classification," *Journal of applied statistics*, vol. 45, no. 15, pp. 2800–2818, 2018.

[135] W. H. Beluch, T. Genewein, A. Nürnberger, and J. M. Köhler, "The power of ensembles for active learning in image classification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 9368–9377.

[136] N. Pawlowski, M. Jaques, and B. Glocker, "Efficient variational bayesian neural network ensembles for outlier detection," *arXiv preprint arXiv:1703.06749*, 2017.

[137] G. S. Tandel, A. Tiwari, and O. G. Kakde, "Performance optimisation of deep learning models using majority voting algorithm for brain tumour classification," *Computers in Biology and Medicine*, vol. 135, p. 104564, 2021.

[138] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," *Advances in neural information processing systems*, vol. 30, 2017.

[139] P. Domingos, "A Few Useful Things to Know About Machine Learning," *Communications of the ACM*, vol. 55, no. 10, pp. 78–87, 2012.

[140] K. You, M. Long, J. Wang, and M. I. Jordan, "How Does Learning Rate Decay Help Modern Neural Networks?" *arXiv preprint arXiv:1908.01878*, 2019.

[141] C. Oinar and E. Kim, "Self-Driving Car Steering Angle Prediction: Let Transformer Be a Car Again," *arXiv preprint arXiv:2204.12748*, 2022.

[142] W. S. Gosset, "The Probable Error of a Mean," *Biometrika*, pp. 1–25, 1908.

[143] X. Wang and K. He, "Enhancing the Transferability of Adversarial Attacks through Variance Tuning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 1924–1933.

[144] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn, "Gradient surgery for multi-task learning," *Advances in neural information processing systems*, vol. 33, pp. 5824–5836, 2020.

[145] **Jingyu Zhang**, Jacky Keung, Xiaoxue Ma, Yihan Liao, Yishu Li and Yicheng Sun, "Enhancing the transferability of adversarial attacks for end-to-end autonomous driving systems," in *31st Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2024.

[146] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *stat*, vol. 1050, p. 20, 2015.

[147] J. Lin, C. Song, K. He, L. Wang, and J. E. Hopcroft, "Nesterov Accelerated Gradient and Scale Invariance for Adversarial Attacks," *arXiv preprint arXiv:1908.06281*, 2019.

[148] J. Zou, Z. Pan, J. Qiu, X. Liu, T. Rui, and W. Li, "Improving the transferability of adversarial examples with resized-diverse-inputs, diversity-ensemble and region fitting," in *European Conference on Computer Vision*. Springer, 2020, pp. 563–579.

[149] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 618–626.

[150] W. Wu, Y. Su, X. Chen, S. Zhao, I. King, M. R. Lyu, and Y.-W. Tai, "Boosting the transferability of adversarial samples via attention," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 1161–1170.

[151] T. Xiang, H. Liu, S. Guo, Y. Gan, W. He, and X. Liao, "Towards query-efficient black-box attacks: A universal dual transferability-based framework," *ACM Transactions on Intelligent Systems and Technology*, vol. 14, no. 4, pp. 1–25, 2023.

[152] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[153] J. Gildenblat, "Visualizations for understanding the regressed wheel steering angle for self-driving cars." 2016, *https://github.com/jacobgil/keras-steering-angle-visualizations*.

[154] A. Staravoitau, "Behavioral cloning: end-to-end learning for self-driving cars." 2016, *https://github.com/navoshta/behavioral-cloning*.

[155] U. Challenge, "Steering angle community model: Cg23." https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models/cg23, 2016.

[156] D.-H. Lee, K.-L. Chen, K.-H. Liou, C.-L. Liu, and J.-L. Liu, "Deep learning and control algorithms of direct perception for autonomous driving," *Applied Intelligence*, vol. 51, no. 1, pp. 237–247, 2021.

[157] S. Hecker, D. Dai, and L. Van Gool, "Failure prediction for autonomous driving," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 1792–1799.

[158] K. T. Co, L. Muñoz-González, L. Kanthan, B. Glocker, and E. C. Lupu, "Universal adversarial robustness of texture and shape-biased models," in *2021 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2021, pp. 799–803.

[159] NVIDIA, "Nvidia-autopilot-keras." https://github.com/0bserver07/Nvidia-Autopilot-Keras, 2016.

[160] **Jingyu Zhang**, Jacky Keung, Xiaoxue Ma, Xiangyu Li, Yan Xiao, Yishu Li, and Wing Kwong Chan, "Enhancing valid test input generation with distribution awareness for deep neural networks," in *IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2024, pp. 1095–1100.

[161] D. J. Hauser and N. Schwarz, "Attentive turkers: Mturk participants perform better on online attention checks than do subject pool participants," *Behavior research methods*, vol. 48, pp. 400–407, 2016.

[162] D. Hendrycks, M. Mazeika, and T. Dietterich, "Deep anomaly detection with outlier exposure," *arXiv preprint arXiv:1812.04606*, 2018.

[163] M. Attaoui, H. Fahmy, F. Pastore, and L. Briand, "Black-box safety analysis and retraining of dnns based on feature extraction and clustering," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 3, pp. 1–40, 2023.

[164] T. Zohdinasab, V. Riccio, A. Gambi, and P. Tonella, "Deephyperion: exploring the feature space of deep learning-based systems through illumination search," in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2021, pp. 79–90.

[165] M. O. Attaoui, H. Fahmy, F. Pastore, and L. Briand, "Supporting safety analysis of image-processing dnns through clustering-based approaches," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 5, pp. 1–48, 2024.

[166] M. Talo, "Automated classification of histopathology images using transfer learning," *Artificial intelligence in medicine*, vol. 101, p. 101743, 2019.

[167] K. Pearson, "Liii. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, vol. 2, no. 11, pp. 559–572, 1901.

[168] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," *arXiv preprint arXiv:1802.03426*, 2018.

[169] E. Bingham and H. Mannila, "Random projection in dimensionality reduction: applications to image and text data," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, 2001, pp. 245–250.

[170] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, vol. 5. University of California press, 1967, pp. 281–298.

[171] R. S. King, "Cluster analysis and data mining: An introduction," 2013.

[172] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[173] L. McInnes, J. Healy, S. Astels *et al.*, "hdbscan: Hierarchical density based clustering." *J. Open Source Softw.*, vol. 2, no. 11, p. 205, 2017.

[174] A. Diaz-Papkovich, L. Anderson-Trocmé, and S. Gravel, "A review of umap in population genetics," *Journal of Human Genetics*, vol. 66, no. 1, pp. 85–91, 2021.

[175] Y. Hozumi, R. Wang, C. Yin, and G.-W. Wei, "Umap-assisted k-means clustering of large-scale sars-cov-2 mutation datasets," *Computers in biology and medicine*, vol. 131, p. 104264, 2021.

[176] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.

[177] D. Moulavi, P. A. Jaskowiak, R. J. Campello, A. Zimek, and J. Sander, "Density-based clustering validation," in *Proceedings of the 2014 SIAM international conference on data mining*. SIAM, 2014, pp. 839–847.

[178] G. Aksu, C. O. Güzeller, and M. T. Eser, "The effect of the normalization method used in different sample sizes on the success of artificial neural network model," *International journal of assessment tools in education*, vol. 6, no. 2, pp. 170–192, 2019.

[179] D. Li and Q. Li, "Adversarial deep ensemble: Evasion attacks and defenses for malware detection," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3886–3900, 2020.

[180] Udacity, "Using Deep Learning to Predict Steering Angles." 2016, *https://github.com/udacity/self-driving-car/tree/master/challenges/challenge-2*.

[181] A. Chernikova, A. Oprea, C. Nita-Rotaru, and B. Kim, "Are self-driving cars secure? evasion attacks against deep neural networks for steering angle prediction," in *2019 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2019, pp. 132–137.

[182] N. Šrndić and P. Laskov, "Practical evasion of a learning-based classifier: A case study," in *2014 IEEE symposium on security and privacy*. IEEE, 2014, pp. 197–211.

[183] L. Schott, J. Rauber, M. Bethge, and W. Brendel, "Towards the first adversarially robust neural network model on mnist," *arXiv preprint arXiv:1805.09190*, 2018.

[184] A. Mądry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *stat*, vol. 1050, no. 9, 2017.

[185] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, “An empirical comparison of model validation techniques for defect prediction models,” *IEEE Transactions on Software Engineering*, vol. 43, no. 1, pp. 1–18, 2016.

[186] X. Zhou, Y. Jin, H. Zhang, S. Li, and X. Huang, “A map of threats to validity of systematic literature reviews in software engineering,” in *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2016, pp. 153–160.

[187] DAVE-dropout, “Behavioral cloning: end-to-end learning for self-driving cars.” 2016. [Online]. Available: https://github.com/alexstaravoitau/behavioral-cloning

[188] Udacity, “Image-based Localization.” 2016, *https://github.com/udacity/self-driving-car/tree/master/challenges/challenge-3*.

[189] C. Wissler, “The spearman correlation formula,” *Science*, vol. 22, no. 558, pp. 309–311, 1905.