MISSIONE 4
ISTRUZIONE
RICERCA

SCUOLA DI DOTTORATO
UNIVERSITÀ DEGLI STUDI DI MILANO-BICOCCA

Testing Deep Learning Systems: From Practice to Prioritization in Autonomous Driving Systems

PhD Thesis

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science

by

Mr. Qurban Ali

Registration No: 906921

Supervisor:	Prof. Leonardo Mariani
Co-supervisor:	Prof. Oliviero Riganelli
Tutor:	Prof. Yuri Pirola
Director PhD Program:	Prof. Gianluca Della Vedova

Department of Informatics, Systems and Communication (DISCo)

University of Milano-Bicocca, Italy

Academic Year **2025/2026**

Finanziato dall'Unione europea NextGenerationEU

Ministero dell'Università e della Ricerca

Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA

UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

Abstract

Deep learning (DL) has become a core enabling technology for Autonomous Driving Systems (ADS), yet systematic testing and validation of such systems remain a major challenge due to their data-driven nature, vast input spaces, and strong dependence on execution context. In particular, existing testing practices, benchmark reuse, and regression testing strategies have not kept pace with the scale and complexity of modern DL-based ADAS. This thesis investigates these challenges through a multi-layered empirical and methodological study, spanning software engineering practices, infrastructure interoperability, and behavior-aware test optimization.

First, the thesis presents a large-scale empirical study of testing practices in open-source Python deep learning projects. By analyzing 300 DL repositories, the study characterizes test adoption, automation, coverage practices, and test suite evolution over time. The results show that, although test adoption increases as projects mature, model-specific tests and non-functional testing remain underrepresented. Moreover, test suites grow substantially as projects evolve, intensifying the need for scalable and efficient regression testing. These findings expose systematic maturity gaps in DL testing and motivate the development of more principled testing methodologies, particularly for safety-critical domains such as autonomous driving.

Second, the thesis investigates the reusability of large-scale ADS benchmarks across heterogeneous simulation platforms. It introduces *OpenCat*, an open-source infrastructure that enables high-fidelity conversion of OpenDRIVE road representations into Catmull-Rom splines, allowing industry-grade benchmarks such as SensoDat to be executed in lightweight academic simulators. While OpenCat achieves near-perfect geometric fidelity across more than 32,000 scenarios, extensive cross-platform evaluation reveals substantial divergence in pass/fail outcomes when scenarios are executed using different Advanced Driver Assistance Systems (ADAS) models. These results expose a fundamental limitation of geometry-centric benchmarking: test outcomes are tightly coupled to the underlying system architecture and execution context, leading to model-specific brittleness that undermines benchmark interoperability and reproducibility.

Finally, the thesis proposes a *behavior-aware* test suite reduction and prioritization framework for ADAS regression testing. The framework combines geometric properties of road segments with dynamic behavioral features extracted from execution traces to cluster, select, and prioritize test scenarios. An extensive evaluation across multiple driving environments and imitation-learning ADAS models demonstrates that the proposed approach reduces test execution cost by up to 89% while retaining the majority of failure-inducing scenarios

and significantly improving early fault detection compared to random, geometric-only, and behavior-only baselines. Cross-model experiments further reveal partial transferability of behavior-informed prioritization, highlighting both generalizable driving difficulty patterns and architecture-dependent vulnerabilities.

Overall, this thesis advances the state of the art in DL-based ADAS testing by (i) empirically characterizing testing maturity in real-world DL projects, (ii) exposing fundamental interoperability and model-dependence limitations of existing benchmarks, and (iii) introducing scalable, behavior-aware testing techniques that improve regression efficiency without sacrificing fault detection capability. The findings underscore the need to move beyond purely geometric and syntactic notions of test adequacy toward behavior-centric and architecture-aware testing methodologies for reliable and reproducible ADS validation.

Acknowledgements

A doctoral thesis is never the work of one individual alone. This dissertation reflects the guidance, support, and generosity of many people who accompanied me through what has been the most demanding and formative period of my life. I am deeply grateful to all of them.

I would like to express my deepest gratitude to my research supervisor, Professor Leonardo Mariani, for his constant intellectual guidance, mentorship, and encouragement. His ability to see the potential in ideas and push them toward clarity and impact has shaped both this work and my development as a researcher. I am equally grateful to my co-supervisor, Professor Oliviero Riganelli, whose critical insight and unwavering support have left a mark on every contribution in this thesis.

I owe special thanks to Dr. Andrea Stocco for making my mobility period at the Technical University of Munich (TUM) and Fortiss both possible and transformative. His mentorship, collaboration, and intellectual generosity were central to the development of the key contributions of this thesis. The experience in Munich broadened my perspective and strengthened my approach to research.

I would also like to sincerely thank the reviewers of this thesis, Professor Barbara RE and Professor Ajitha Rajan, for their valuable feedback and insightful comments, which helped improve the quality and clarity of this work.

My sincere appreciation goes to my colleagues at the Laboratory of Testing and Analysis (LTA). The collaborative spirit, openness, and daily exchanges, both academic and personal, made the lab a place of growth and belonging. I am equally grateful to my colleagues at TUM and Fortiss for their welcoming and intellectually enriching community.

My sincere thanks also go to the PNRR Centro Nazionale HPC, whose funding made this work possible. This Doctoral thesis was carried out within the framework of the ISCS project funded by the PNRR Mission 4 Component 2 Investment 1.4, funded by the European Union – NextGenerationEU – CUP H43C22000520001

To my family, I owe more than words can express. My mother's unwavering support and belief in me have been the foundation of this journey. To my father and the rest of my family, thank you for your patience, encouragement, and trust in a path that was often uncertain.

Lastly, I thank the friends with whom I shared university life. Their companionship, support, and humor provided balance during the most challenging moments, reminding me of the importance of life beyond research.

Contents

I Foundations and Landscape **10**

1 Introduction **11**

1.1 The Rise of Deep Learning in Safety-Critical Systems . . . 11
1.2 The Assurance Challenge for ADS . . . 12
1.3 The Testing Challenges: Gaps in Practice, Interoperability, and Efficiency . 13
1.3.1 Inadequate Testing Practices in DL Projects . . . 13
1.3.2 The Interoperability Bottleneck in ADS Testing . . . 15
1.3.3 The Test Suite Explosion: Redundancy and Prioritization in ADS . 16
1.4 Contributions . . . 18
1.4.1 Empirical Insight into DL Testing Practices: . . . 18
1.4.2 OpenCat: An Interoperability Bridge for ADS Testing: . . . 19
1.4.3 Coverage-Guided Test Reduction and Prioritization Framework: . . 19
1.5 Thesis Structure . . . 19

2 Background and Key Concepts **21**

2.1 Fundamentals of Software Testing . . . 21
2.1.1 Test Levels . . . 21
2.1.2 Test Types . . . 22
2.1.3 Regression Testing: Test Minimization, Selection, and Prioritization 24
2.2 Deep Learning Systems: Characteristics and Testing Challenges . . . 26
2.2.1 Data Dependency . . . 26
2.2.2 Non-Determinism . . . 27
2.2.3 Lack of Interpretability . . . 27
2.2.4 High-Dimensional and Continuous Input Spaces . . . 27
2.2.5 Challenges for Testing and Quality Assurance . . . 28
2.3 Autonomous Driving Systems (ADS) and ADAS Testing . . . 29
2.3.1 Architecture of an ADS . . . 30
2.3.2 Simulation-based testing for ADS . . . 31
2.3.3 Lane-Keeping ADAS as a Case Study . . . 35
2.4 Key Technologies and Techniques . . . 36
2.4.1 Road Representation Formats . . . 37
2.4.2 Dynamic Time Warping (DTW) . . . 40

3 From DL Testing to ADAS Regression Testing **43**

3.1 Testing Practices in Open-Source Software Projects . . . 43

3.1.1 General OSS Testing Practices . . . 43
3.1.2 Implications for Deep Learning Projects . . . 44
3.1.3 Testing Practices in ML/DL Projects . . . 44
3.2 Testing and Validation of ADS . . . 45
3.2.1 Benchmarks for ADS Testing . . . 46
3.3 ADAS Regression Testing . . . 50
3.3.1 Search-Based Approaches . . . 51
3.3.2 Scenario-Based Approaches . . . 51
3.3.3 Clustering-Based Approaches . . . 52
3.4 Interoperability and Format Conversion in ADS Testing . . . 53
3.4.1 Interoperability . . . 54
3.4.2 Format Conversion Tools . . . 56
3.5 Identified Research Gaps . . . 57
3.5.1 Limited Empirical Understanding of DL Testing Practices . . . 57
3.5.2 Benchmark Portability and Model-Specific Brittleness . . . 58
3.5.3 Scalable Regression Testing for DL-Based ADAS . . . 58

II Empirical Investigation and Core Contributions 59

4 Understanding Testing Practices in Deep Learning Projects 60
4.1 Testing of Deep Learning Projects . . . 60
4.2 Study Design . . . 62
4.2.1 Research Questions . . . 62
4.2.2 Project Selection . . . 63
4.3 Results and Analysis . . . 70
4.3.1 RQ_{1b}(Test Case Presence and Types) . . . 70
4.3.2 RQ_{1b}(Test Automation Adoption) . . . 72
4.3.3 RQ_{1c}(Code Coverage Analysis) . . . 73
4.3.4 RQ_{1d}(Test Suite Evolution and Maintenance) . . . 74
4.4 Discussion . . . 77
4.4.1 Promising Trends . . . 77
4.4.2 Critical Gaps . . . 79
4.5 Threats to Validity . . . 80

5 OpenCat: An Interoperability Bridge for ADS Testing 83
5.1 The Scalability Challenge of ADS Benchmark . . . 83
5.1.1 The Interoperability Challenge: Format Lock-in . . . 83
5.1.2 The Semantic Challenge: The Model-Specific Brittleness . . . 85
5.2 The OpenCat Conversion Approach . . . 85

5.2.1 System Overview 85
5.2.2 Conversion Algorithm 86
5.3 Experimental Evaluation 89
5.3.1 RQ_{2a} (Accuracy): Quantitative and Qualitative Validation 90
5.3.2 RQ_{2b} (Comparison): Pass/Fail Ratio 91
5.4 Discussion and Implications 94
5.4.1 Pass/Fail Differences 95
5.4.2 OpenCat as an Enabler for Cross-Platform Testing 96
5.5 Threats to Validity 97

6 Coverage-Guided Road Selection and Prioritization for Accelerated Testing of Autonomous Driving Systems 100
6.1 Scalability Challenge 100
6.2 Behavior-aware Prioritization 102
6.3 Framework Architecture 104
6.3.1 Segmentation and Curvature Analysis of Roads 106
6.3.2 Computation of Geometric Distances 109
6.3.3 Computation of Dynamic Distances 111
6.3.4 Clustering of Road Sections 112
6.3.5 Test Selection 114
6.3.6 Test Prioritization 116
6.4 Empirical Evaluation 119
6.4.1 Research Questions 119
6.4.2 Objects of Study 120
6.4.3 Experimental Platforms and Benchmarks 121
6.4.4 Procedure and Metrics 122
6.4.5 Results 125
6.5 Threats to Validity 136

III Synthesis and Conclusion 139

7 Conclusion and Future Work 140
7.1 Key Contributions 140
7.1.1 Empirical Characterization of Deep Learning Testing Practices . . . 140
7.1.2 OpenCat: Infrastructure for Cross-Platform Benchmark Reuse . . . 141
7.1.3 Coverage-Guided Prioritization Framework 142
7.2 Implications for Research and Practice 143
7.2.1 Model-Specific Testing as a Necessity 143
7.2.2 Behavioral Fidelity as a Coverage Dimension 143

7.2.3 Context Coupling and Benchmark Interpretation 143
7.2.4 Scalability Requires Selection, Not Exhaustion 143
7.3 Limitations of the Presented Work 144
7.3.1 Scope and Generalizability Constraints 144
7.3.2 Interoperability and Transfer Limitations 144
7.3.3 Prioritization Framework Constraints 145
7.3.4 Simulation-to-Reality Gap 146
7.4 Future Research Directions 147
7.4.1 Extending Empirical Studies of Testing Practice 147
7.4.2 Advancing Behavioral Benchmarking 147
7.4.3 Improving Test Selection and Cold-Start Prioritization 147
7.4.4 Closing the Simulation-to-Reality Gap 148
7.4.5 Extending to Complex ADAS Functions and Additional Simulator Formats 148
7.5 Closing Remarks 148

List of Figures

2.1 Autonomous driving system overview [1]. 32
2.2 Road representation formats. 39

4.1 TensorFlow test suite growth. 75
4.3 PyTorch test suite growth. 76
4.2 Keras test suite growth. 76

5.1 Conversion process from OpenDRIVE format to Catmull-rom spline. . . . 86
5.2 Comparative view of OpenDRIVE road, generated spline, and test case. . . 88

6.1 Coverage-based road selection and prioritization framework. 105
6.2 Illustrative example of the proposed framework 119
6.3 Comparison between geometric-only and hybrid approaches (Dave-2). . . . 129
6.4 Comparison between geometric-only and hybrid approaches (Chauffeur). . 130
6.5 Cross-model test failure consistency analysis 132
6.6 Failure retention across models (Dave-2/Chauffeur). 132

List of Tables

1.1 SAE levels of driving automation [2] 13

2.1 High-level comparison of autonomous driving simulators. 33

4.1 Distribution of stars, forks, and contributors per framework 65
4.2 List of the selected GitHub repositories 69
4.3 Test occurrence in the selected projects 71
4.4 Type of test cases implemented in Python GitHub projects 71
4.5 Test automation frameworks 72
4.6 Distribution of code coverage tools 73
4.7 Distribution of projects with code coverage percentage 74
4.8 DL framework commit analysis 77

5.1 Test pass/fail counts across campaigns. 92

6.1 Framework configuration parameters 106
6.2 Coverage-based selection and prioritization-DAVE-2. 126
6.3 Coverage-based selection and prioritization-Chauffeur. 127
6.4 Failure detection comparison between DAVE-2 and Chauffeur models. . . . 135

Part I

Foundations and Landscape

Chapter 1

Introduction

1.1 The Rise of Deep Learning in Safety-Critical Systems

Deep Learning (DL) has fundamentally transformed software engineering, enabling systems to learn complex patterns from data rather than relying solely on explicitly programmed rules [3]. Driven by advances in computational power, the availability of large-scale datasets, and breakthroughs in neural architecture design, DL models can now achieve nearly human-level performance in tasks such as image classification, natural language processing, and speech recognition [4, 5]. This paradigm shift has led to the widespread adoption of DL technologies in safety-critical domains, including autonomous driving, medical diagnosis, aviation, and industrial automation [6, 7, 8, 9], where failures can result in loss of life, environmental damage, or significant economic harm, and introduce profound challenges for software quality assurance [10, 11].

Unlike traditional software systems, DL systems exhibit emergent behavior that depends on training data, model architecture, and learned parameters such as weights and biases across millions of neurons, which complicate their validation [12]. This data-driven paradigm introduces unique challenges for software quality assurance. While conventional software can be tested through extensive exploration of discrete input/output mappings and deterministic execution paths, deep learning models operate in high-dimensional continuous spaces where even small, imperceptible input perturbations can cause severe misclassifications [5]. Consequently, a model that performs well on test data from the training distribution may fail catastrophically when exposed to out-of-distribution inputs [13]. Moreover, the inherent lack of interpretability in deep neural networks complicates defect localization and root-cause analysis, exacerbating their brittleness in safety-critical applications where unpredictable environments and rare edge cases can lead to catastrophic consequences [14, 15].

Consider ADS, where DL models perform critical perception tasks such as object detection, semantic segmentation, and depth estimation, as well as control functions like steering and speed regulation [16]. Operating in open-world environments characterized by infinite variability: changing weather, diverse road geometries, unpredictable traffic behavior, and sensor noise [17], these systems face immense challenges in ensuring safety and reliability. A single perception error, such as misclassifying a pedestrian as a shadow, or a control failure in navigating a sharp curve, can lead to fatal consequences [18, 19].

Such failures are not limited to autonomous driving; similar risks extend to other safety-critical domains such as aviation and healthcare, where incorrect decisions by AI-driven systems can have life-threatening consequences [7, 8]. These applications face rigorous scrutiny as regulatory bodies demand testing and validation approaches that go beyond traditional accuracy metrics, emphasizing robustness, generalization, interpretability, and worst-case performance under adversarial or out-of-distribution conditions [20, 7].

1.2 The Assurance Challenge for ADS

ADS represent one of the most complex and safety-critical applications of DL technology. Modern ADS integrate multiple DL components, including cameras, LiDAR, radar, and ultrasonic sensors for perception, prediction, and planning, creating a system that must operate reliably in highly dynamic and unpredictable environments with minimal or no human intervention [16, 17, 8]. The assurance of these systems presents extraordinary challenges that extend beyond conventional software validation [21, 22].

The Society of Automotive Engineers (SAE) defines six levels of driving automation, ranging from Level 0 (no automation) to Level 5 (full automation under all conditions) [23] as presented in Table 1.1. Currently, most commercially deployed systems predominantly operate at Level 2 (partial automation), where ADAS handle steering and acceleration but still require constant human supervision [24]. Higher levels of automation (3-5) remain largely experimental, facing significant technical, regulatory, and safety challenges [20, 14]. Despite this limited autonomy, Level 2 systems have been linked to numerous accidents, revealing critical shortcomings in their testing and validation [25]. According to the U.S. National Highway Traffic Safety Administration (NHTSA), 392 crashes involving ADAS were reported between July 2021 and May 2022, with 273 incidents attributed to systems employing Level 2 automation [25]. These statistics highlight that even *assisted* driving systems, which ostensibly require human oversight, exhibit failure modes that escape pre-deployment testing.

The safety assurance of ADS is challenged by the infinite variability of real-world scenarios, ranging from road geometry and weather conditions to traffic patterns, pedestrian behavior, and sensor degradation [23]. Unlike traditional embedded systems, which have well-defined operational boundaries and deterministic behavior, ADS must operate reliably under rare, high-risk edge cases and adversarial conditions that may not be represented in the training data. This inherent complexity, coupled with real-time performance requirements and the probabilistic nature of deep learning components, makes traditional testing methodologies insufficient for validating ADS reliability, which relies on explicit requirements and predictable outputs [26, 27].

Simulation-based testing has emerged as a cornerstone of ADS validation, allowing de-

Table 1.1: SAE levels of driving automation [2]

Level	Name	Description	DDT Performer
0	No Driving Automation	Driver performs all aspects of the dynamic driving task (DDT) even when enhanced by warning or intervention systems	Human driver
1	Driver Assistance	System provides sustained steering OR acceleration/deceleration support; driver performs all other DDT aspects	Human driver
2	Partial Driving Automation	System provides sustained steering AND acceleration/deceleration support; driver monitors environment and performs remainder of DDT	Human driver
3	Conditional Driving Automation	System performs all DDT within operational design domain (ODD); driver must be ready to take over when requested	System (with fallback-ready user)
4	High Driving Automation	System performs all DDT within ODD; no driver intervention required, even if the user does not respond to takeover request	System
5	Full Driving Automation	System performs all DDT under all roadway and environmental conditions that can be managed by a human driver	System

velopers to safely and efficiently evaluate vehicle behavior across thousands of synthetic scenarios that would be impractical, dangerous, and prohibitively expensive to execute in the real world [28]. High-fidelity simulators such as CARLA [29], BeamNG.tech [30], and other commercial platforms accurately model sensor physics, vehicle dynamics, and traffic interactions, providing a controlled and reproducible environment for stress-testing. However, the effectiveness of this approach critically depends on the quality, diversity, and representativeness of the generated scenarios, as well as the fidelity of the simulation environment itself [31]. As scenario volume scales with systematic test generation and benchmark datasets, new challenges arise in managing test redundancy, execution time, and the interpretability of benchmarks across the different simulation platforms [22, 32].

1.3 The Testing Challenges: Gaps in Practice, Interoperability, and Efficiency

This thesis identifies and addresses challenges that hinder the effective testing and validation of DL-based systems, particularly in the context of ADS. These challenges emerged progressively through our research, each motivating the subsequent phase of investigation.

1.3.1 Inadequate Testing Practices in DL Projects

Despite the critical role of testing in ensuring software quality, empirical evidence about actual testing practices in DL projects remained largely anecdotal [33]. While software engineering has well-established norms for testing, such as achieving high code coverage,

automating test execution, and maintaining comprehensive regression suites, it remains unclear to what extent these practices translate to DL projects [34, 35, 36]. DL systems differ fundamentally from traditional software because their behavior is not explicitly programmed; instead, it is learned from data [4]. As a result, correctness is tied to data distributions and training dynamics rather than deterministic control flow or logic structures [3]. This means that conventional coverage metrics (e.g., statement or branch coverage) provide little insight into whether the learned model behavior has been sufficiently exercised [37].

Furthermore, DL systems introduce testing challenges that traditional software practices do not directly address. For example, a perception model used in autonomous driving may perform well under clear daylight conditions but fail under fog or nighttime lighting, even though no code has changed [38]. Such failures arise not from implementation errors but from distribution shift between training and operational data. This scenario illustrates that ensuring correctness in DL systems requires evaluating robustness and generalization across diverse environmental conditions, considerations that are largely absent from traditional software testing [5].

To establish a solid empirical foundation for our research agenda, we conducted the first comprehensive investigation of testing practices across 300 open-source DL projects hosted on GitHub, utilizing popular frameworks such as TensorFlow, PyTorch, and Keras (detailed in Chapter 4). To investigate testing practices within Python DL projects, we formulated the following research question:

- **$\mathbf{RQ_1}$ (Testing Practices):** *What are the current testing practices in deep learning projects, and what gaps exist, particularly for model-specific testing and test evolution?*

We investigated the presence and evolution of test suites, the types of tests implemented (unit, integration, system), the adoption of test automation, the use of code coverage metrics, and the integration of testing into CI/CD workflows.

This study revealed both promising trends and critical gaps. While 77% of projects include some form of test suite, only 55% implement tests specifically targeting DL model components, indicating that developers tend to validate surrounding infrastructure rather than the models themselves. Functional testing is the most common type of testing performed in these projects, and unit tests are the most frequently used level at which such tests are implemented. In contrast, non-functional aspects such as performance and security testing are significantly underrepresented. Moreover, only 23% of projects reported code coverage metrics, and coverage of model-specific code was often incomplete. This discrepancy indicates that current testing practices may insufficiently address the core learned behavior of deep learning models. Further investigation is required to understand the causes of this gap and to evaluate how effectively these projects validate model

reliability in practice. Although 86% adopted automation frameworks (e.g., PyTest, Unittest), only one-third integrated automated execution into CI/CD pipelines. This gap is especially concerning for DL systems, where models evolve frequently through retraining and data updates. Without CI/CD integration, regressions in model behavior may remain undiscovered for long periods, slowing development cycles and increasing the risk of deploying degraded or unsafe models [39]. We also observed rapid growth in test suites (up to 10×), suggesting that test maintenance and regression management will become increasingly important as DL systems evolve.

While RQ_1 addresses DL testing practices broadly, its findings have direct implications for safety-critical DL applications such as autonomous driving systems (ADS). The empirical observation that test suites grow rapidly (up to 10×) while model-specific validation remains limited raises an urgent question for the ADS domain, where inadequate testing can have life-threatening consequences. Furthermore, the identified gaps in coverage reporting and test automation adoption suggest that even well-established DL projects may lack the testing infrastructure necessary for rigorous regression testing. These insights motivated a focused investigation into ADS testing, during which we identified two challenges: benchmark interoperability (addressed in 1.3.2) and test suite scalability (addressed in 1.3.3). Specifically, we investigate whether these benchmarks possess sufficient generality to support cross-model regression testing for meaningful reuse.

1.3.2 The Interoperability Bottleneck in ADS Testing

ADS is a domain where testing complexity is exacerbated by ecosystem fragmentation. The ADS testing landscape comprises numerous simulation platforms, road representation formats, and benchmark datasets that remain largely incompatible. Benchmarks such as SensoDat [40], DeepScenario [41], and SCTrans [42] offer thousands of curated road scenarios for regression testing. However, these benchmarks are often tightly coupled to specific simulators and ADAS models, limiting their portability and re-usability. Portability is crucial in ADS testing, as it enables comparing different models, validating system updates, and reproducing results across platforms [43, 44, 45].

For instance, SensoDat provides 32,580 scenarios in the OpenDRIVE format, intended for execution within BeamNG.tech using its built-in PID-based controller. A PID (Proportional-Integral-Derivative) controller computes steering corrections based on lane deviation. Its tuned control dynamics shape how the vehicle behaves on a given road, meaning that pass/fail outcomes are influenced not only by the scenario but also by controller-specific behavior. While OpenDRIVE provides detailed geometric and semantic road descriptions, supporting complex road networks, lane structures, intersections, and elevation profiles [46], its complexity and tight coupling to specific simulators restricts cross-platform use. Conversely, many academic research simulators, such as Udacity's Self-

Driving Car Simulator [47], rely on simpler geometric representations like Catmull–Rom splines, favoring efficiency but sacrificing representational richness such as lane markings, intersection logic, and traffic infrastructure. This gap motivates the following research question:

- **RQ_2 (Interoperability):** *Are existing ADS benchmarks truly reusable across different simulators and ADAS models, or do they exhibit platform-specific brittleness?*

To bridge this interoperability gap, we developed OpenCat, the first open-source converter that transforms OpenDRIVE road representations into Catmull-Rom spline representations (detailed in Chapter 5). Applying OpenCat to the complete SensoDat dataset, we achieved perfect geometric fidelity, confirming that technical conversion is feasible. We then re-executed the converted scenarios using an independent DL-based lane-keeping ADAS model (DAVE-2) within the Udacity simulator. Interestingly, while SensoDat originally reported a 61% pass rate in BeamNG.tech, our re-execution produced a 98% pass rate, revealing pronounced model-specific brittleness, where scenario outcomes depend more on the ADAS architecture than on scenario difficulty.

These findings highlight that the improvements were not due to conversion artifacts, which were ruled out through geometric validation. Instead, they point to a conceptual challenge: model-specific behaviors may influence performance, implying that benchmarks designed for one ADAS model and simulator may not generalize effectively to architecturally distinct models, thereby limiting their usefulness for cross-model regression testing.

This realization has two major implications. First, it challenges the fundamental assumption that benchmarks can serve as universal test suites for ADS. Second, it highlights the urgent need for test selection and prioritization methods, given that the converted OpenCat dataset comprised over 32,000 road scenarios, and executing the full suite would require an impractically large amount of simulation time. This scalability challenge underscored the need to identify and prioritize test cases that capture diverse and representative driving scenarios. It became the foundation of our third research challenge and motivated the development of our coverage-driven test selection and prioritization framework for large-scale test suite optimization.

1.3.3 The Test Suite Explosion: Redundancy and Prioritization in ADS

With OpenCat enabling cross-platform testing and producing an interoperable, large-scale dataset of 32,580 scenarios drawn from multiple generation campaigns, a new practical challenge emerges: the rapid growth of test suite size and the prohibitive cost of exhaustive execution. Executing the full converted dataset requires more than 1,000 hours of continuous simulation, and this cost increases substantially when multiple ADAS architectures must be validated or when frequent regression testing is performed, quickly

rendering exhaustive replay impractical [48, 32].

At the same time, the OpenCat dataset contains substantial behavioral redundancy. Many scenarios, despite geometric variation, induce nearly identical vehicle trajectories and control responses, contributing little additional validation value. Indiscriminate execution of such redundant tests not only wastes computational resources but also delays the discovery of critical failures by allocating effort to scenarios that are unlikely to reveal new faults [49]. For example, road segments that differ only marginally in curvature often produce indistinguishable driving behavior, making multiple executions unnecessary. Identifying and eliminating these redundancies, therefore, offers the opportunity to significantly reduce testing cost, without sacrificing fault-detection capability.

To address this scalability challenge, we developed a coverage-guided test reduction and prioritization framework (detailed in Chapter 6) that jointly considers geometric and behavioral coverage metrics to retain diverse driving scenarios. This ensures that both typical and challenging road scenarios remain represented even as the test suite is reduced.

Our approach decomposes roads into homogeneous sections (straight, left-curve, right-curve), compares them using Dynamic Time Warping (DTW) [50] to detect geometric similarity, and integrates dynamic metrics extracted from simulation logs, including speed variability, steering variability, cross-track error, and yaw rate, to capture behavioral patterns beyond geometry alone. These combined features drive hierarchical agglomerative clustering [51] to group similar road segments while preserving geometric and behavioral diversity. From these clusters, we select a minimal yet representative subset of test cases and prioritize them based on a multi-criteria scoring function combining geometric complexity, dynamic difficulty, and historical failure information. To evaluate the effectiveness of our test selection and prioritization methods, we formulated the following research question:

- **$\mathbf{RQ_3}$ (Test Reduction and Prioritization):** *How can we efficiently reduce and prioritize regression test suites for ADAS while maintaining fault detection capability and generalizing across different models?*

We evaluated the framework using two architecturally distinct ADAS models: DAVE-2 [52], a feedforward convolutional neural network with five convolutional and three fully connected layers, and Chauffeur [53], which employs a deeper convolutional structure with six layers (dropout and pooling) and critically integrates recurrent LSTM layers to process temporal context. This architectural diversity enables rigorous assessment of cross-model generalization.

Empirical evaluation demonstrated that our framework achieves substantial test suite reduction (up to 89%) for both models, reducing execution time from approximately 1,000 hours to fewer than 100 hours per ADAS version. At the same time, it retains most

failure-inducing scenarios (up to 75%) and improves early fault detection by up to 95× faster than random test orderings, consistently across the DAVE-2 and Chauffeur models. Furthermore, Hybrid selection combining geometric and dynamic features substantially outperformed either feature set in isolation (geometric-only/dynamic-only), improving fault detection by 60-85% depending on campaign type and architecture; while geometric-only and dynamic-only configurations each identified only 3-25% of failures, confirming that behavioral features meaningfully refine geometric clustering [19].

To assess cross-model generalization, we evaluate whether test cases prioritized for one model can also reveal failures in the other. Specifically, we measure the overlap of failure-inducing test cases between the Chauffeur and DAVE-2 models. Our results show meaningful but incomplete transfer across architectures: the test cases selected for DAVE-2 retain 70% of Chauffeur failures, while those selected for Chauffeur retain 74% of DAVE-2 failures. These findings indicate that behavior-aware test selection captures generalizable driving difficulty patterns while also exposing architecture-specific vulnerabilities [32].

To address these challenges, this thesis systematically examines key limitations in DL testing for ADS, spanning current testing practices, the interoperability and robustness of test benchmarks, and the scalability of regression testing techniques. The first goal is to *empirically characterize testing practices in deep learning projects*, particularly focusing on model-specific testing and test suite evolution patterns (RQ_1), through a large-scale empirical analysis of open-source DL projects hosted on GitHub. The second goal is to *investigate benchmark interoperability and model-specific brittleness in ADS testing*, examining the extent to which test cases that expose failures in one model transfer to other models, and assessing whether existing test benchmarks can be reused across different simulators and ADAS models (RQ_2). The third goal is to *develop a test suite reduction and prioritization framework* to improve testing efficiency while preserving behavioral and geometric diversity, maintaining fault detection capability, and ensuring generalization (RQ_3).

1.4 Contributions

This thesis makes the following scientific and practical contributions:

1.4.1 Empirical Insight into DL Testing Practices:

We provide the first comprehensive characterization of testing practices in Python DL projects, revealing critical gaps in model-specific testing, coverage reporting, and utilization of test automation tools. Most importantly, we document intense test suite maintenance activity and rapid growth, highlighting the urgent need for test management strategies in safety-critical domains. The research presented above has been conducted in collaboration

with other researchers and has resulted in the following peer-reviewed publication:

- Qurban Ali, Oliviero Riganelli, and Leonardo Mariani. "Testing in the Evolving World of DL Systems: Insights from Python GitHub Projects." In *Proceedings of the 24th IEEE International Conference on Software Quality, Reliability and Security (QRS)*, 2024.

1.4.2 OpenCat: An Interoperability Bridge for ADS Testing:

We introduce OpenCat, the first open-source tool for converting OpenDRIVE road representations to Catmull-Rom splines. By applying OpenCat to the SensoDat dataset (32,580 scenarios), we obtain 100% geometric accuracy in the derived scenarios. Our cross-platform evaluation reveals a difference in pass/fail ratios when using an independent ADAS model, exposing limitations in benchmark generalizability. A detailed account of this work appears in the following publication:

- Qurban Ali, Andrea Stocco, Leonardo Mariani, and Oliviero Riganelli. "OpenCat: Improving Interoperability of ADS Testing." In *Proceedings of the 47th IEEE/ACM International Conference on Software Engineering Workshops (ICSEW)*, 2025.

1.4.3 Coverage-Guided Test Reduction and Prioritization Framework:

Leveraging the OpenCat converted dataset, we propose a novel framework for test suite reduction and prioritization that addresses both scalability (reducing test count) and brittleness (generalizing across models). This contribution provides a practical solution for regression testing that balances efficiency with effectiveness. The contribution outlined above has led to the following publication:

- Qurban Ali, Leonardo Mariani, Andrea Stocco, and Oliviero Riganelli. "Coverage-Guided Road Selection and Prioritization for Efficient Testing in Autonomous Driving Systems." In *Proceedings of the 33rd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2026.

1.5 Thesis Structure

The remainder of this thesis is organized as follows. Chapter 2 provides background on software testing fundamentals, DL system characteristics, ADS architecture, and the key analytical techniques underlying our contributions. Chapter 3 reviews related work on testing practices, ADS validation, interoperability, and test prioritization, positioning our work within the broader research landscape. Chapters 4, 5, and 6 present the three core contributions: (1) characterization of testing practices in Python DL projects, (2) OpenCat: a tool for converting the OpenDRIVE road representation to Catmull-Rom

splines, and (3) a novel coverage-guided test reduction and prioritization framework, respectively. Chapter 7 concludes with a summary of contributions, limitations, and future research directions.

Chapter 2

Background and Key Concepts

This chapter provides the foundational knowledge necessary to understand the research presented in the subsequent chapters. We begin by reviewing the fundamental concepts in software testing, characterizing DL systems and their distinguishing properties, followed by an overview of the ADS architecture and simulation-based testing. Finally, we introduce the key analytical techniques, including road representation formats and DTW, establishing the mathematical and practical foundations that underpin our empirical investigations in the preceding Chapters 3, 4, and 5.

2.1 Fundamentals of Software Testing

Software testing is a systematic process of evaluating a software application to detect bugs, verify that it meets specified requirements, and assess its quality before deployment [54]. Testing serves multiple purposes: it validates functional correctness, uncovers hidden bugs, ensures system reliability, and provides confidence that the software behaves as expected under diverse conditions. In modern software engineering, testing is integrated throughout the Software Development Life Cycle (SDLC), enabling early defect detection and reducing the cost of fixing bugs discovered late in the development process [55].

2.1.1 Test Levels

Test levels define the *scope* and *granularity* at which testing is performed, ranging from fine-grained checks of individual components to end-to-end validation of the entire system [56, 57]. Each level serves a distinct purpose and targets different classes of defects.

2.1.1.1 Unit Testing

Unit testing validates individual software units, typically functions, methods, or classes, in isolation from the rest of the system [56]. These tests are closely tied to the source code and form the first line of defense against defects. Because unit tests are fast to execute and relatively inexpensive to maintain, they are well-suited for integration into continuous integration (CI) pipelines using frameworks such as PyUnit or *UnitTest* [57, 58].

In DL systems, unit tests often focus on utility methods (e.g., data pre-processing, feature normalization), configuration parsing, and loss or metric implementations, but they rarely exercise the end-to-end model behavior itself [59]. This creates a gap: even when unit test

coverage is high, misbehaviors can still emerge from learned representations or data issues that are not directly captured by code-level tests [12].

2.1.1.2 Integration Testing

Integration testing verifies that multiple components or modules work correctly when combined, focusing on interfaces, data exchange, and control flow between units [56]. Unlike unit tests, which often use mocks or stubs, integration tests rely on real interactions between components, making them more resource-intensive but also more effective at exposing interface mismatches and protocol violations [57].

In DL-enabled ADS, integration tests typically ensure that the data loader correctly feeds batched sensor data to the model, that perception outputs (e.g., detected lanes or objects) are correctly consumed by planning and control modules, and that pre- and post-processing steps (e.g., coordinate transformations) remain consistent across updates [44]. Failures at this level can lead to subtle but critical defects, such as misaligned coordinate frames or misinterpreted confidence scores, which may not be visible in isolated unit tests.

2.1.1.3 System Testing

System testing evaluates the complete, integrated software system against its specified functional and non-functional requirements [56, 57]. These tests emulate realistic end-user interactions or operational workflows, validating end-to-end behavior across all components. System tests are typically more expensive to execute and maintain, but provide the most holistic view of system quality.

For ADS, system testing entails evaluating the entire driving stack, including perception, prediction, planning, and control over complete driving scenarios in high-fidelity simulation or test tracks [60, 16]. Such scenarios may cover diverse road geometries, traffic interactions, weather conditions, and rare edge cases, making system tests the primary vehicle for assessing overall safety and robustness [61]. However, their high cost and complexity motivate the need for scalable scenario selection and prioritization strategies, as investigated in this thesis.

2.1.2 Test Types

While test levels describe *where* testing is applied in the software architecture, test types describe *what* aspects of quality are being assessed (e.g., functional correctness, performance, robustness) [57]. Multiple test types can be applied at each level, depending on the testing objectives and constraints.

2.1.2.1 Functional Testing

Functional testing evaluates a software system against its functional requirements and specifications, verifying that it produces correct outputs for given inputs and behaves as expected in defined use cases [54]. Functional tests can be implemented at different levels: unit-level checks for individual functions, integration tests for module interactions, and system-level tests for end-to-end workflows.

In DL/ADS contexts, functional testing includes verifying that:

- Pre-processing and post-processing steps (e.g., image normalization, coordinate transformations) behave as specified.
- Models produce outputs within expected ranges or satisfy invariants (e.g., steering angles remain bounded on straight roads).
- ADS pipelines respect high-level rules (e.g., stopping at red lights, yielding to pedestrians) under nominal conditions [44].

Because full functional specifications are rarely available for DL models, functional testing often relies on partial specifications, example-based expectations, oracles derived from baselines, or metamorphic relations [62, 63].

2.1.2.2 Non-Functional Testing

Non-functional testing assesses quality attributes not directly tied to specific functional outputs, such as performance, scalability, robustness, and security [57]. For DL/ADS, non-functional requirements are crucial: even if the system produces correct decisions, violations of timing, robustness, or security constraints can still lead to unsafe behavior [64].

Performance Testing: Performance testing evaluates how the system behaves under expected and peak workloads, focusing on metrics such as throughput (e.g., frames per second processed), latency (e.g., inference time per frame), and resource utilization (e.g., CPU/GPU usage, memory consumption) [57]. For DL models deployed in ADS, strict real-time constraints apply: perception and control loops must complete within tens of milliseconds to ensure timely reactions to dynamic environments [16]. Performance testing, therefore, verifies that end-to-end pipelines meet latency bounds across different hardware configurations and that performance degrades gracefully under increased load (e.g., additional sensors, higher traffic density). Violations of timing constraints can result in stale decisions, delayed braking, or unstable control, even when the underlying model predictions are correct [64].

Security Testing: Security testing examines how a system behaves under abnormal, noisy, or malicious conditions [65]. For deep learning (DL) systems, this includes robustness to distribution shift, sensor noise, corrupted inputs, and adversarial perturbations [12]. In ADS, security testing often involves simulating degraded sensing conditions (e.g., rain, fog, partial occlusions), unusual road geometries, or rare corner-case scenarios to ensure that model performance degrades safely rather than catastrophically [44, 66].

Beyond traditional software vulnerability analysis, security testing for DL-based ADAS must also address ML-specific threats, such as adversarial examples, data poisoning, and model extraction attacks [67, 68]. For instance, small, carefully crafted perturbations to road textures or traffic signs can induce misclassification or unsafe control decisions without producing obvious visual artifacts [69]. While this thesis does not propose new robustness or security testing methods, the empirical study presented in Chapter 4 demonstrates that such non-functional tests are rarely implemented in open-source DL projects, despite their critical importance in safety-critical domains.

Smoke Testing: Smoke testing is performed to ascertain the fundamental functionality of an application at a very high level [70]. It provides early assurance that critical system features operate as expected. Additionally, they are useful for assessing newly built software and ensuring that no major issue exists by verifying proper functionality following deployment to a fresh environment [57, 71].

Overall, distinguishing between test levels and test types clarifies how different testing activities contribute to the assurance of DL-enabled ADS. Unit, integration, and system tests provide complementary coverage across the architecture, while functional and non-functional tests address correctness and critical quality attributes such as performance, robustness, and security. This thesis builds on these foundations to study how testing is practiced in DL projects (Chapter 4) and how scenario-based testing for ADS can be scaled and prioritized effectively (Chapters 5 and 6).

2.1.3 Regression Testing: Test Minimization, Selection, and Prioritization

Building upon the fundamental testing concepts introduced in the previous section, we now examine specific strategies for managing test suites as software systems evolve, a critical concern for continuous integration and deployment practices [72]. Regression testing is the process of re-executing existing test cases after software modifications to verify that previously working functionality remains intact and that no new defects have been introduced [73]. As software systems evolve through multiple releases, their test suites tend to expand rapidly, making exhaustive execution increasingly expensive and time-consuming [74]. Thus, effective test suite management is required to balance thorough validation against resource constraints. To address this challenge, three main strategies

are employed: test minimization, selection, and prioritization (explained below). These strategies help maintain regression testing effectiveness while reducing execution cost [73].

2.1.3.1 Test Minimization

Test minimization (also called test suite reduction) aims to remove redundant test cases from the test suite, while preserving essential coverage and fault-detection capability [75]. Tests are considered redundant when they exercise the same program paths, verify identical functionality, or provide overlapping fault-detection capability compared to other tests in the suite. For example, if two test cases achieve identical code coverage and detect the same set of faults, one can be safely removed without loss of testing effectiveness [76]. In the context of ADS, geometric redundancy occurs when multiple road scenarios induce nearly identical vehicle trajectories and control responses despite superficial differences in road layout [77]. However, test minimization introduces inherent risks: permanently discarded tests cannot detect faults in newly modified or added code, potentially reducing long-term fault-detection capability as the system evolves [78]. Studies show that minimized test suites may miss up to 50% of regression faults compared to the original suite when code undergoes significant changes [78]. Therefore, minimization strategies must carefully balance reduction effectiveness against the risk of weakening regression testing power.

2.1.3.2 Test Selection

Test selection aims to reduce the cost of regression testing by temporarily selecting a subset of test cases relevant to recent code changes, where cost is measured in terms of both the number of selected test cases and total execution time [79]. Unlike minimization, which permanently removes tests, selection creates execution subsets tailored to specific modifications while retaining the full suite for comprehensive validation when needed [73]. Selection criteria typically involve analysis of modified program entities (i.e., functions, classes, and modules), tracing which tests exercise those entities, and selecting only the affected tests [76]. While minimization seeks a permanently reduced suite based on redundancy, selection dynamically adapts to changing context; the same test may be selected for one modification but excluded for another [73]. This flexibility enables fast feedback loops in continuous integration while maintaining the option for exhaustive testing during major releases [80].

Traditional selection techniques rely on static code analysis, which assumes deterministic program behavior [12]. However, DL systems introduce fundamental complications: (1) model behavior is data-driven rather than explicitly programmed, making it difficult to predict which tests are affected by weight updates [12], and (2) small parameter changes can produce non-local behavioral shifts affecting seemingly unrelated test cases [81]. Consequently, DL-based systems require selection strategies grounded in behavioral similarity

and input space coverage rather than code-level change analysis [82]. While test selection is crucial for the efficiency and fast feedback loops in the development process, it introduces the risk of reducing fault detection effectiveness if critical tests are excluded during the selection process [79].

2.1.3.3 Test Prioritization

Test prioritization reorders test cases to maximize early fault detection, ensuring that the most valuable tests execute first without removing any tests from the suite [79]. Unlike minimization and selection, prioritization preserves complete coverage while optimizing execution order to concentrate fault-revealing tests at the beginning of the sequence, enabling faster feedback and earlier detection of critical defects [83]. Prioritization strategies may be coverage-based (prioritizing tests covering frequently modified or rarely covered code), fault-based (prioritizing tests historically associated with defect detection), risk-based (prioritizing tests exercising high-risk functionality), or hybrid approaches combining multiple criteria [79, 83].

2.2 Deep Learning Systems: Characteristics and Testing Challenges

Deep learning (DL) systems differ fundamentally from traditional software. Instead of encoding behavior through explicit rules and deterministic control logic, DL models learn their behavior from data through iterative optimization [34, 35]. A DL system typically consists of a neural network architecture (layers, activation functions, connections), a training dataset, an optimization algorithm, and an inference pipeline [3]. This shift from rule-based to data-driven behavior means that correctness is no longer determined solely by code, but by complex interactions among data, model architecture, and training dynamics, making it much harder to reason about expected behavior, anticipate failure modes, and define complete test oracles [84, 64]. As a result, the specific properties of DL systems significantly complicate testing and quality assurance compared to traditional software [12].

2.2.1 Data Dependency

A DL model's behavior is determined by patterns learned from training data rather than explicit programmer-defined logic and data structures. Consequently, DL correctness depends on the quality, diversity, and representativeness of training datasets. Unlike traditional software, where correctness can sometimes be specified and verified against formal requirements or exhaustive logical conditions, DL systems rarely have complete, executable specifications of desired behavior across the full input space [84, 12]. Instead,

their *specification* is implicit in the data and loss function, which may omit rare but safety-critical corner cases. When inputs during deployment differ from training data, known as *distribution shift*, model performance can degrade unpredictably, leading to silent failures that are difficult to anticipate with traditional test design [85]. This makes it essential that test data reflect realistic operational conditions and cover rare but critical situations, yet collecting and curating such data (e.g., long-tail driving events) is often costly, time-consuming, and constrained by privacy or safety concerns [16].

2.2.2 Non-Determinism

DL systems are inherently stochastic: different training runs may yield different parameterizations even with identical data, architecture, and hyperparameters, due to random initialization, non-deterministic GPU execution, and stochastic optimization procedures [84]. This non-determinism can lead to variations in accuracy, robustness, and failure patterns across model instances that nominally implement the same design, complicating reproducibility and regression testing, retraining *the same* model may change which inputs fail and why [86]. For safety-critical systems, such variability raises concerns about the stability of validation results over time and across deployments.

2.2.3 Lack of Interpretability

DL models, particularly deep convolutional and recurrent networks, operate as high-dimensional black boxes with millions or billions of parameters and complex internal representations. Understanding why a model made a specific prediction or took a particular action is challenging, as there is no simple, human-readable mapping between internal parameters and functional behavior [64, 87]. This lack of interpretability complicates debugging and root-cause analysis: when a failure occurs, it is difficult to determine whether the cause lies in the data (e.g., bias, label error), the model (e.g., overfitting, spurious correlations), or the environment (e.g., sensor noise, occlusions) [27]. It also hinders the reproducibility of failure investigations, since different teams or tools may produce different explanations for the same behavior. Consequently, testing DL systems often requires additional instrumentation (e.g., saliency maps, feature attribution) and specialized analysis to gain partial insight into internal decision processes [12].

2.2.4 High-Dimensional and Continuous Input Spaces

Unlike traditional software, which often operates on discrete and enumerable inputs, DL models consume continuous, high-dimensional data such as images, LiDAR point clouds, or multi-sensor time series. The resulting input space is effectively infinite, making exhaustive testing infeasible even for relatively simple tasks [84]. Moreover, DL models can be highly sensitive to small, carefully crafted input perturbations, known as *adversarial examples*,

that are often imperceptible to humans but cause the model to produce incorrect or unsafe outputs [88, 67]. For instance, adding small, structured noise to a traffic sign image can cause a classifier to mislabel a stop sign as a speed-limit sign, or slightly altering roadside textures may cause a lane-keeping model to drift out of its lane [69]. In ADS, such adversarial vulnerabilities can manifest as missed obstacles, incorrect lane detection, or unsafe steering actions, posing a serious challenge for safety assurance [64, 66]. This combination of high-dimensionality and adversarial sensitivity makes it difficult to design test suites that meaningfully cover the operational domain and robustly probe worst-case behaviors.

2.2.5 Challenges for Testing and Quality Assurance

Traditional software testing focuses on techniques such as code coverage, deterministic test oracles, and regression analysis based on code changes [82]. These principles do not transfer directly to DL systems, where behavior is encoded in learned parameters and evolves through retraining rather than code modification. For example, two different models may share identical source code but diverge significantly in behavior due to differences in training data or random initialization, rendering code-based coverage metrics insufficient as proxies for behavioral coverage [12]. As a result, DL-specific testing techniques have emerged, including metrics and methods that explicitly target neuron activation patterns, input transformations, robustness, and model perturbations [89, 62]. Common DL-oriented techniques include:

- **Neuron coverage:** Inspired by code coverage, neuron coverage measures how many neurons (or activation patterns) are exercised by a test suite, with the intuition that higher coverage corresponds to greater behavioral diversity [81, 82]. Variants such as k-multisection neuron coverage and top-k neuron coverage refine this idea by considering activation ranges and intensity.

- **Metamorphic testing:** Metamorphic testing defines *metamorphic relations*, expected relationships between inputs and outputs under certain transformations, to serve as *relational* test oracles when exact outputs are unknown [62, 63]. For example, in image classification, rotating or slightly translating an object should not change its label; in ADS, adding light rain or small illumination changes should not cause grossly different steering commands.

- **Adversarial testing:** Adversarial testing systematically generates perturbed inputs (e.g., via gradient-based attacks or search-based methods) to expose robustness weaknesses and identify decision boundaries where the model is fragile [88, 68]. In ADS, this includes generating subtle modifications to road textures, lighting, or object placement that cause misperceptions or unsafe control decisions [66].

- **Mutation testing:** DL mutation testing perturbs model parameters, architectures, or training data (e.g., weight noise, neuron removal, label noise) to create *mutant* models and evaluate whether existing tests can detect behavioral deviations [82, 90]. A high mutation score indicates that the test suite is sensitive to changes in model behavior and thus more adequate.

These techniques aim to approximate behavioral coverage, define practical oracles in the absence of full specifications, and stress-test robustness under realistic or adversarial perturbations [12]. However, despite promising results in research prototypes, adoption remains limited in practice. Our empirical study of 300 DL projects (Chapter 4) shows that although 77% of the projects include some form of test suite, only 55% of them include tests directly targeting model behavior, and non-functional testing (e.g., robustness, performance, security) remains significantly underrepresented. This gap between available techniques and industrial practice motivates the need for DL-aware testing frameworks that are both effective and practically integrable into existing development workflows [27, 91].

2.3 Autonomous Driving Systems (ADS) and ADAS Testing

ADS require extensive testing to ensure safe and reliable operation. This typically combines simulation-based verification and real-world validation. Verification focuses on determining whether the system satisfies its design requirements, while validation assesses whether the system performs robustly under realistic driving scenarios [16, 17]. The scale of this challenge is considerable: realistic operational design domains (ODDs) span vast combinations of road geometries, traffic densities, weather conditions, sensor configurations, and rare edge-case events, making purely real-world testing prohibitively expensive and time-consuming [92, 44]. This thesis focuses on ADAS and, within that category, on a single control-level function: lane-keeping assistance. This is an SAE Level 1-2 capability that maps camera input directly to steering commands, operating without multi-agent negotiation, explicit path planning, or full environment perception.

Regulatory developments further increase the urgency and stringency of robust testing. Emerging standards and guidelines, such as ISO 26262 (functional safety) [93], ISO/PAS 21448 (SOTIF) [94], ISO/SAE 21434 (cybersecurity) [95], and the UNECE Global Technical Regulation on ADS safety [96], explicitly require evidence that ADS have been validated against a representative and risk-aware set of scenarios before deployment [96, 44]. These regulations emphasize scenario-based testing, traceability from requirements to tests, and systematic coverage of safety-relevant situations, raising the bar for both the breadth and depth of ADS testing.

In this context, simulation plays a central role by enabling large-scale, repeatable, and controllable experimentation that would be infeasible or unsafe to perform on public

roads [16, 31]. However, the increasing availability of large scenario datasets and benchmarks also leads to test-suite explosion, interoperability challenges across simulators and models, and the need for principled test selection and prioritization strategies (detailed in Chapters 5 and 6). The remainder of this section introduces the ADS architecture and simulation-based testing workflow that underpin these challenges and motivate the contributions of this thesis.

2.3.1 Architecture of an ADS

ADS are complex Cyber Physical Systems (CPS) that integrate hardware and software components, including sensors, perception, planning, and control, to enable vehicles to navigate the environment with varying levels of human intervention as reported in Table 1.1 [17, 97]. As shown in Figure 2.1, modern ADS typically follow a modular architecture with the following stages:

Sensors: Sensors provide the raw data required for an autonomous vehicle to understand its surroundings and its own speed and position. Exteroceptive sensors, such as cameras, LiDAR, and radar, capture information about external objects and road geometry. Proprioceptive sensors, including GPS and inertial measurement units (IMU), estimate the vehicle's position and orientation. The data from these heterogeneous sensors is fused and forwarded to the perception module, which constructs a coherent representation of the driving environment [97, 98].

Perception: The perception module processes inputs from onboard sensors and external sources, such as high-definition maps. Its role is to detect and classify objects (vehicles, pedestrians, traffic signs), estimate their positions and velocities, identify lane markings, and segment drivable areas [99]. Deep learning models, particularly Convolutional Neural Networks (CNNs), are widely used in this module because of their ability to learn hierarchical visual features [100]. These models support key perception tasks including object detection and tracking, lane detection, and semantic segmentation [97].

Planning: The planning module generates safe, efficient trajectories that navigate the vehicle safely from its current position to the destination while obeying traffic rules, avoiding obstacles, and ensuring passenger comfort [98]. Planning is typically decomposed into route planning (high-level path from A to B), prediction (forecasting the future trajectories of detected objects), behavioral planning (lane-change decisions, merging), and motion planning (low-level trajectory generation) [97].

Control: The control module translates planned trajectories into actuator commands (steering angle, throttle, and brake) that execute the desired vehicle motion. Controllers

such as Proportional-Integral-Derivative (PID) and Model Predictive Control (MPC) are commonly used, though learning-based controllers are increasingly explored, as they can learn from real-time input-output data and are particularly useful for systems where accurate models are difficult to obtain. [101, 97].

This modular architecture approach offers advantages for testing and maintainability, such as well-defined component boundaries and the ability to isolate faults. However, modularity also introduces challenges. Errors can propagate across modules; for instance, a perception failure (e.g., misclassifying a pedestrian) may lead to cascading failures in prediction, planning, and control. Effective testing must therefore account for such inter-component dependencies [11, 101].

An alternative to modular architectures, *End-to-End Learning* employs a single deep neural network that maps raw sensor inputs directly to control commands, bypassing explicit intermediate representations. NVIDIA's DAVE-2 model [101] exemplifies this approach by learning to drive from human demonstration data. Although end-to-end approaches simplify the system architecture by sacrificing interpretability and modularity, thereby complicating debugging, safety analysis, and verification, as the network does not produce interpretable intermediate outputs [3, 101]. If a failure occurs (e.g., the vehicle crashes), it is often unclear whether the failure stems from inadequate perception, poor trajectory planning, or incorrect control. The absence of explicit intermediate representations makes fault attribution substantially more difficult [102].

Despite these challenges, end-to-end models have demonstrated strong performance in controlled and simulated environments [101, 103] and they are increasingly being investigated in real-world driving contexts [104, 102]. They continue to attract research interest because they offer a conceptually simple, unified pipeline from raw sensor input to control, reduce the need for costly hand-engineered perception and planning modules, and can exploit large-scale data to learn complex, non-linear driving policies that are difficult to encode manually [104, 102]. This thesis focuses specifically on testing end-to-end lane-keeping models, including Dave-2 [101] and Chauffeur [53], as they represent a canonical testbed for investigating testing challenges in deep learning-based ADS.

2.3.2 Simulation-based testing for ADS

Simulation-based testing is the cornerstone of ADS verification and validation (V&V). Relaying solely on physical road testing is insufficient and impractical for several reasons [105, 106]. First, testing unverified ADS on public roads poses risks to passengers, pedestrians, property, and surrounding vehicles. Moreover, physical testing is resource-intensive, expensive, requiring dedicated vehicles, trained safety drivers, instrumentation, and access to diverse driving environments, making large-scale testing costly and slow.

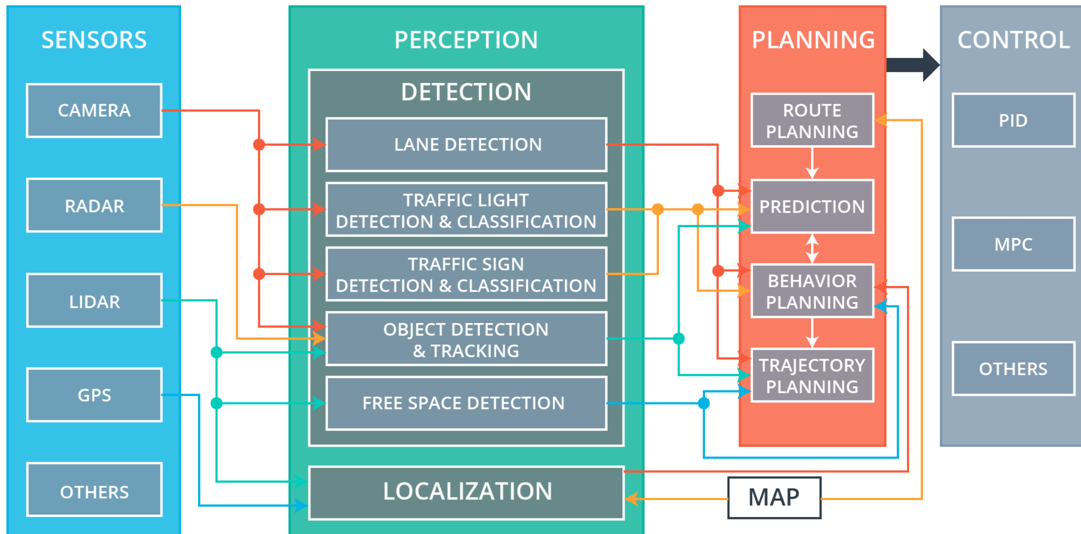


Figure 2.1: Autonomous driving system overview [1].

Real-world testing also suffers from limited scenario coverage. Many safety-critical situations, such as sudden pedestrian crossings, sensor malfunctions, or rare adverse weather conditions, may occur infrequently and cannot be reliably or repeatedly encountered on demand[66]. Finally, real-world conditions lack strict reproducibility; even small variations in lighting, traffic density, or road conditions can impede consistent debugging and regression testing [107].

High-fidelity simulators such as *CARLA* [29], *BeamNG.tech* [30], and *LGSVL* [108] address these limitations by providing controlled, reproducible virtual environments that are difficult or risky to obtain in the real world. These simulators model sensor physics, vehicle dynamics, traffic behavior, and environmental conditions with varying degrees of realism, enabling rigorous evaluation of perception, planning, and control components under diverse scenarios [29, 106]. In contrast, lightweight environments such as the *Udacity* [47] and *Donkey* [109] simulators are lower in fidelity but particularly useful for rapid prototyping and experimentation with end-to-end models [102]. Table 2.1 presents the detailed comparison between these simulators.

Simulation-based testing typically follows structured workflows. First, *test scenarios are generated* by specifying road geometries, traffic configurations, weather conditions, and initial vehicle states. Scenarios can be manually designed, extracted from real-world logs, or systematically generated using search-based and combinatorial methods [110]. Next, *simulation execution* executes the ADAS model in the virtual environment while recording sensor data, predicted actions, and resulting vehicle trajectories [111]. Finally, the system's behavior is assessed through *test oracles*, which provide automated criteria for determining

Simulator	Fidelity	Dynamics	Sensors & perception	Traffic / interaction	Typical use
CARLA	High	Urban driving simulator; supports maps/scenarios.	Configurable sensor suites and conditions.	Scenario-based actors/traffic support.	Research training/validation/benchmarking.
BeamNG.tech	High	Soft-body physics; detailed vehicle dynamics.	Simulated sensors + automation APIs.	Scenario-based testing environments.	ADAS testing + dynamics-centric evaluation.
LGSVL	High	Full-stack AD simulation (SIL-style workflows).	Configurable sensors; stack integration.	Controllable objects/digital-twin focus.	Integration with autonomy stacks (e.g., Autoware/Apollo).
Udacity	Low-Med	Unity-based educational simulator.	Camera-driven end-to-end prototyping.	Limited interactive traffic vs. high-fidelity stacks.	Rapid prototyping/teaching for end-to-end models.
Donkey	Low	Lightweight platform for fast iteration.	Supports NN/CV/GPS autopilots; sim-to-hardware workflow.	Track-based setups; interaction depends on the environment.	Fast experimentation (end-to-end / RL-style control).

Table 2.1: High-level comparison of autonomous driving simulators.

whether a scenario execution should be considered successful or failing. Common oracles include: [106]

- **Out-of-Bound (OOB) Detection:** The vehicle leaves the drivable road surface.
- **Collision Detection:** The vehicle collides with obstacles or other agents such as pedestrians or other vehicles.
- **Rule Violation:** The ADAS violates traffic rules (e.g., running a red light).

In this work, we adopt the OOB detection oracle, as it provides a clear, simulator-independent, and widely used criterion for identifying critical failures in lane-keeping tasks, where maintaining lane boundaries is the primary functional objective. Collision- and rule-based oracles are not considered, as the evaluated scenarios do not involve dynamic agents or traffic control elements, and introducing such oracles would require additional assumptions about environment semantics and agent behavior beyond the scope of this study.

Prior work [28, 112] has empirically studied the transferability of simulation-based testing to real-world vehicles, showing that while simulation can uncover many real-world failures, some failure modes only manifest in physical environments. Consequently, simulation-based testing should be viewed as a complement to, rather than a replacement for, real-world validation.

2.3.2.1 Udacity Self-Driving Car Simulator

The Udacity Self-Driving Car Simulator [47] is an open-source simulation platform developed using the Unity 3D game engine [113] with the NVIDIA PhysX physics engine [114]. The simulator provides:

Tracks: Two pre-built tracks (Lake Track and Jungle Track) featuring roads with varying curvature, elevation, and scenery.

Custom Roads: Users can define custom road geometries using Catmull-Rom splines specified as sequences of waypoints.

Sensor Simulation: Front-facing camera providing RGB images at 160×320 resolution, vehicle telemetry (speed, steering angle, throttle, brake), and cross-track error (CTE), which measures lateral deviation from the lane centerline.

Autonomous Mode: The simulator exposes a control interface where external ADAS models (e.g., Dave-2) can receive camera images and send steering/throttle commands via a socket connection.

This thesis uses the Udacity simulator for all empirical evaluations (Chapters 5 and 6). This choice is motivated by several practical considerations. The simulator provides native support for end-to-end ADAS models such as Dave-2 [52] and Chauffeur [53], enabling direct integration, real-time inference, and built-in telemetry collection. It also supports programmatic road generation via Catmull–Rom splines, which is essential for injecting and executing thousands of benchmark scenarios at scale. Furthermore, its lightweight Unity-based architecture enables efficient large-scale experimentation without the computational overhead of high-fidelity simulators. Finally, its widespread adoption in ADAS testing research [115, 19, 60] facilitates reproducibility and comparison with prior work.

2.3.2.2 BeamNG.tech Simulator

BeamNG.tech [30] is a high-fidelity vehicle simulation platform originally developed for video gaming but increasingly used for automotive research and testing. Key features include:

Soft-Body Physics: Unlike rigid-body simulators, BeamNG models vehicles as soft bodies with deformable components, enabling realistic crash simulations and vehicle dynamics.

High Visual Fidelity: Advanced rendering techniques (ray tracing, physically-based materials) produce visually realistic scenes.

Extensive Scenarios: BeamNG supports complex urban environments, diverse weather conditions, and dynamic traffic.

OpenDRIVE Support: BeamNG natively imports road networks defined in the OpenDRIVE format, facilitating integration with industry-standard scenario specifications.

The SensoDat benchmark [40], which we investigate in Chapter 5, was created using BeamNG.tech. BeamNG's AI driver (a PID-based controller with global knowledge of the road topology) was used to label scenarios as *pass* or *fail* based on whether the vehicle

remained within lane boundaries. Our research explores whether these labels generalize to different ADAS models tested in different simulators (e.g., Udacity).

2.3.3 Lane-Keeping ADAS as a Case Study

Lane-keeping assistance (LKA) is a canonical Level 2 ADAS capability that uses vision-based perception to detect lane markings and outputs steering commands to maintain the vehicle within lane boundaries [101]. At inference time, the model receives real-time camera images and predicts steering angles to keep the vehicle centered in the lane. The challenge lies in generalizing to roads not seen during training, including sharp curves, narrow lanes, and complex geometries [116, 117].

Lane-keeping ADAS is selected as the primary use case for this thesis on deliberate methodological grounds. First, it provides experimental control: as a single-agent, single-function task with a well-defined pass/fail oracle (out-of-bounds detection), lane-keeping isolates the testing methodology from task-level complexity. Introducing multi-agent interactions, pedestrian intent modelling, or additional ADAS functions such as Automated Emergency Braking (AEB), Blind Spot Monitoring (BSM), or Adaptive Cruise Control (ACC) would conflate framework validity with scenario complexity, making it difficult to attribute observed testing behaviour to the methodology rather than the task. Extension to multi-agent ADAS functions is discussed as future work in Section 7.4. Second, lane-keeping is among the most extensively studied ADS tasks in the regression testing literature [118, 119, 19, 115], providing a rich set of baselines and benchmarks, including SensoDat, comprising 32,580 curated scenarios, against which novel contributions can be rigorously evaluated and compared. Third, the task is supported by mature, open-source infrastructure (DAVE-2, Chauffeur, Udacity simulator, SensoDat) that enables large-scale empirical evaluation without the prohibitive cost of constructing new simulation environments from scratch.

2.3.3.1 Dave-2 Architecture

Dave-2 is a convolutional neural network architecture developed by NVIDIA for end-to-end learning of driving behavior [101]. The model takes as input a $(160 \times 320 \times 3)$ RGB image representing a road scene. The network consists of five convolutional layers: three with (5×5) kernels (24, 36, and 48 filters respectively, each with a stride 2), followed by two layers with (3×3) kernels (64 filters each). These layers learn hierarchical visual representations relevant for driving. The convolutional backbone is followed by three fully connected layers with 100, 50, and 10 neurons, culminating in a single output neuron that predicts the steering angle. ReLU activation functions are applied throughout, except at the output layer. Due to its architectural simplicity, strong empirical performance, and publicly available pre-trained implementations, Dave-2 has become a widely used

benchmark model in ADAS testing research [118, 119, 19, 115].

2.3.3.2 Chauffeur Architecture

Chauffeur [53], provides an alternative end-to-end lane-keeping architecture that differs from Dave-2 by incorporating temporal modeling capabilities. It includes six convolutional layers with batch normalization, dropout, and max-pooling for robust spatial feature extraction, followed by two Long Short-Term Memory (LSTM) layers that capture temporal dependencies across sequential video frames. The output of the recurrent layers is passed through fully connected layers that produce the final steering command, enabling the model to leverage both spatial and temporal driving cues [120]. Chauffeur's recurrent architecture allows it to learn smoother trajectories by considering past frames, potentially improving performance on roads with gradual curvature changes [121]. However, this architectural difference also means that Chauffeur and Dave-2 may fail on different road scenarios, a phenomenon we investigate in Chapter 6.

These models were selected based on the following criteria:

- **Architectural diversity:** Dave-2 is a pure feedforward CNN (5 convolutional + 3 fully connected layers) that processes single frames independently, while Chauffeur incorporates recurrent LSTM layers that capture temporal dependencies across sequential frames. This fundamental architectural difference enables rigorous assessment of whether our testing and prioritization methods generalize across distinct model families.
- **Community adoption:** Both models are among the most widely used benchmark subjects in ADS testing research. Dave-2, based on NVIDIA's seminal end-to-end driving architecture[101], has been adopted in numerous testing studies [118, 119, 19, 115]. Chauffeur, developed by the Udacity self-driving community [53], represents a standard recurrent alternative.
- **Reproducibility:** Both models have publicly available implementations and pre-trained weights, enabling reproducible experimentation and comparison with prior work.
- **Complementary failure profiles:** Their architectural differences (feedforward vs. recurrent, single-frame vs. temporal context) produce distinct failure patterns on the same road scenarios, making them ideal for evaluating cross-model generalization of test selection and prioritization strategies.

2.4 Key Technologies and Techniques

In this section, we present the key methods and tools used in our research, with a focus on road representation standards and DTW.

2.4.1 Road Representation Formats

Roads define the spatial trajectories that vehicles must navigate and, therefore, serve as the fundamental test inputs for ADS [122]. However, roads can be represented in multiple formats, each offering different trade-offs in terms of portability, expressiveness, and ease of generation. This thesis specifically investigates OpenDRIVE (an industry-standard format used by the SensoDat benchmark) and Catmull-Rom splines (used by the Udacity simulator), as their incompatibility creates the interoperability challenge addressed by our OpenCat converter in Chapter 5, enabling cross-platform scenario reuse while preserving geometric fidelity. This section describes two key formats: OpenDRIVE and Catmull-Rom splines used in our research [46].

2.4.1.1 OpenDRIVE

OpenDRIVE is an XML-based standard developed by the Association for Standardization of Automation and Measuring Systems (ASAM) for modelling road networks in simulation environments [123]. It supports complex road features, including multi-lane roads, junctions, elevation profiles, lane markings, and traffic signs, as illustrated in Figure 2.2a. Due to this rich expressiveness, OpenDRIVE has become a widely used industry standard for describing road networks in ADAS/ADS development and testing [46].

From a geometric perspective, an OpenDRIVE road is defined by a *reference line* (centerline) and additional profiles (e.g., lane offsets, elevation) specified along the road length [123]. The reference line is parameterized by the arc-length coordinate $s \in [0, L]$ and can be expressed as a planar parametric curve:

$$\mathbf{r}(s) = \begin{bmatrix} x(s) \\ y(s) \end{bmatrix}, \qquad s \in [0, L] \tag{2.1}$$

where L is the length of the geometry element. OpenDRIVE constructs the reference line as a concatenation of geometry primitives (e.g., `line`, `arc`, `spiral`, `paramPoly3`) placed sequentially along s [123]. Each primitive defines how the heading angle $\theta(s)$ evolves along the curve, and the position can be recovered by integrating the unit tangent:

$$\theta(s) = \theta_0 + \int_0^s \kappa(u)\, du, \qquad \mathbf{r}(s) = \mathbf{r}_0 + \int_0^s \begin{bmatrix} \cos\theta(u) \\ \sin\theta(u) \end{bmatrix} du, \tag{2.2}$$

where $\mathbf{r}_0 = (x_0, y_0)$ and θ_0 are the start position and heading of the geometry element, and $\kappa(s)$ is curvature [123].

The main geometry primitives can be described compactly in terms of curvature $\kappa(s)$ [123]:

- **Line:** constant heading, zero curvature: $\kappa(s) = 0$.

- **Arc:** constant curvature (circular arc): $\kappa(s) = \kappa_0 \quad \text{(constant)}$.
- **Spiral (clothoid/Euler spiral):** curvature varies linearly with arc length: $\kappa(s) = \kappa_0 + \dot{\kappa}\, s$

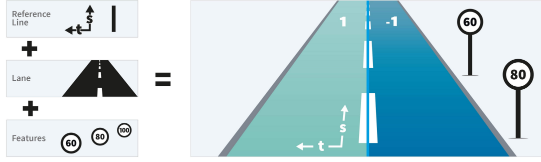
  where κ_0 is the start curvature and $\dot{\kappa}$ is the curvature rate chosen so that the spiral reaches the target end curvature at $s = L$ [123]. This primitive is used to ensure smooth transitions (no curvature jumps) between straight lines and circular arcs, yielding more realistic steering demands.
- **Parametric cubic polynomials:** OpenDRIVE also supports polynomial-based primitives (paramPoly3) for describing curves with parametric cubic functions, enabling additional flexibility in representing complex shapes beyond basic line/arc/spiral compositions [123].

Equation 2.2 clarifies why OpenDRIVE is powerful but also complex: the road shape is defined by a piecewise parametric construction in arc length, where each segment type imposes a specific curvature evolution [123]. This representation is central to this thesis because OpenDRIVE is the native format of industrial benchmarks such as SensoDat. Its expressive geometry must be preserved when translating scenarios into spline-based simulators (Chapter 5) and when extracting curvature profiles used later for segmentation, similarity matching, and clustering (Chapter 6).

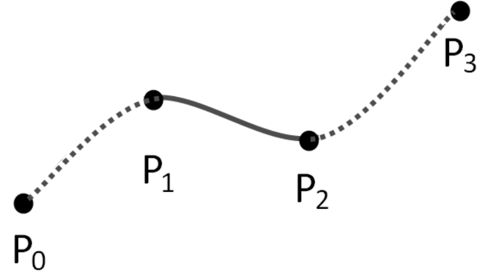
2.4.1.2 Catmull-Rom Spline

A Catmull–Rom spline is a class of cubic interpolation curve widely used in computer graphics, animation, and path generation to create smooth transitions between points [124]. Unlike OpenDRIVE, which represents roads using explicit geometric primitives and rich semantic attributes, Catmull–Rom splines define the road centerline directly by interpolating control points, producing continuous curves without sharp angles or abrupt direction changes [125]. This representation is particularly well-suited for lightweight simulators and enables rapid generation and execution of road-based scenarios in research settings.

Building on these properties, we adopt Catmull–Rom splines as the target representation in OpenCat due to their ability to preserve geometric precision, with conversion achieving 100% accuracy ($R^2 = 1$) while maintaining curvature profiles critical for evaluating steering behavior. Their simplicity allows seamless integration with lightweight academic simulators that lack OpenDRIVE support, facilitating end-to-end evaluation of deep learning-based ADAS models without heavy dependencies. At the same time, they support efficient large-scale experimentation and integrate naturally with imitation learning frameworks such as NVIDIA's DAVE-2 [101], which rely on smooth, continuous road representations.



(a) Elements of OpenDRIVE [123].



(b) Catmull-Rom spline interpolation with four points.

Figure 2.2: Road representation formats.

More broadly, Catmull–Rom splines are widely adopted across graphics and simulation frameworks [124, 126, 127, 113, 128]. Unlike Bézier curves [129] and B-splines [130], they interpolate all control points, ensuring that converted road waypoints lie exactly on the resulting geometry. This property is essential for preserving geometric fidelity and simplifying scenario authoring and validation.

From a mathematical point of view, Catmull-Rom splines compute the curve segment between two consecutive control points by using a cubic polynomial influenced by neighboring points. Given a sequence of control points P_0, P_1, P_2, P_3, the Catmull-Rom spline segment between P_1 and P_2 is computed using the cubic Hermite interpolation formula as defined:

$$C(t) = \frac{1}{2}[(2P_1) + (-P_0 + P_2)t + (2P_0 - 5P_1 + 4P_2 - P_3)t^2 + (-P_0 + 3P_1 - 3P_2 + P_3)t^3] \tag{2.3}$$

where

- $C(t)$ is the position on the time t.
- P_0, P_1, P_2, P_3 are control points.
- t ranges from 0 to 1, interpolating between P_1 and P_2.

This equation ensures that at $t = 0$, $C(t) = P_1$, and at $t = 1$, $C(t) = P_2$, meaning the spline passes exactly through the inner control points P_1 and P_2.

Catmull-Rom splines offer *local control*; modifying one control point only affects the neighboring segments. Combined with their computational efficiency and smoothness properties, this makes them convenient for generating and executing road scenarios in academic simulators such as Udacity [131, 132].

2.4.2 Dynamic Time Warping (DTW)

DTW is a standard algorithm for measuring similarity between two temporal or sequential data of potentially different lengths, producing a distance metric that reflects how well one sequence can be warped to align with another [50]. Unlike simple metrics such as Euclidean distance (which require fixed-length sequences), DTW allows flexible, non-linear alignment between sequences, making it particularly suited for comparing road geometries that may vary in length but exhibit similar curvature patterns [133]. DTW is widely adopted in speech recognition (e.g., comparing audio signals with different speaking rates), gesture recognition, financial time series analysis, and trajectory clustering. In our work (Chapter 6), we apply DTW to compare *curvature profiles* of road segments to identify geometrically similar segments that are likely to induce similar driving behaviors.

Given two curvature sequences $X = [x_1, x_2, \ldots, x_N]$ (curvature values from road segment A) and $Y = [y_1, y_2, \ldots, y_M]$ (curvature values from road segment B), DTW computes an optimal *alignment path* that matches elements of X to elements of Y while minimizing cumulative local distance. The key insight is that the algorithm allows *many-to-one* and *one-to-many* matchings: if road A has a gentle curve and road B has a sharper curve of similar overall shape, DTW can *stretch* or *compress* the alignment to match them, rather than requiring point-by-point correspondence [50].

Mathematically, DTW constructs a cost matrix $D(i, j)$ where entry $D(i, j)$ represents the cumulative cost of the best alignment between the first i elements of X and the first j elements of Y:

$$D(i, j) = d(x_i, y_j) + \min\{D(i-1, j), D(i-1, j-1), D(i, j-1)\} \tag{2.4}$$

where:

- $d(x_i, y_j)$ is the *local distance* between elements at position i in X and position j in Y (typically Euclidean distance: $d(x_i, y_j) = |x_i - y_j|$),
- $D(i-1, j)$, $D(i-1, j-1)$, and $D(i, j-1)$ represent alternative continuations of the alignment path from neighboring cells.

The recurrence relation ensures that the algorithm explores all possible alignments and selects the path with minimum cumulative cost. The *final DTW distance* is $\text{DTW}(X, Y) = D(N, M)$, which represents the optimal alignment cost between the full sequences [50]. Moreover, the DTW distance is a *non-negative value* ($\text{DTW} \geq 0$) with no fixed upper bound [50]:

- $\text{DTW}(X, Y) = 0$ if and only if the two sequences are identical (element-by-element).

- Larger DTW values indicate greater dissimilarity. The magnitude depends on the range and magnitudes of the curvature values, the lengths of the sequences, and the specific alignment path chosen.

- Normalization: To make DTW distances comparable across roads of different lengths, it is common to normalize by sequence length: $\text{DTW}_{\text{norm}}(X, Y) = \frac{\text{DTW}(X,Y)}{N+M}$ or $\frac{\text{DTW}(X,Y)}{\max(N,M)}$ [50].

In Chapter 6, we employ DTW to compare road segments based on their geometric evolution rather than their absolute length. To ensure fair comparison across segments of varying extents, DTW distances are normalized by the combined length of the compared sequences. This normalization allows road segments with similar curvature profiles, such as a long, gradual curve and a shorter but sharper curve, to be recognized as geometrically similar from a *control perspective*, as both impose comparable steering demands and vehicle dynamics [134]. Unlike Euclidean distance on fixed-length samples or simple geometric descriptors (e.g., maximum curvature), DTW's non-linear alignment captures similarities in curvature trajectories while penalizing genuinely different geometries [50]. We apply this normalized DTW measure to compute pairwise distances between all road segments in the OpenCat dataset (Section 3.4.2), forming a distance matrix that is subsequently used for hierarchical agglomerative clustering. This process enables the identification of redundant segments and the selection of a minimal yet representative test suite, supporting efficient and effective ADAS regression testing [134].

This chapter established the foundational knowledge necessary for understanding the research contributions presented in subsequent chapters. We first reviewed core software testing concepts such as test levels and test reduction techniques (minimization, selection, prioritization) to establish the traditional software testing context from which our DL-specific methods depart (Chapter 4). We then characterized DL systems and their distinguishing properties, including data dependency, non-determinism, continuous evolution, and emergent behavior, explaining why these properties complicate conventional testing approaches. Next, we described the architecture of ADS and introduced lane-keeping ADAS models, specifically Dave-2 and Chauffeur, which serve as our primary case study models in this thesis. We also discussed simulation-based testing platforms (Udacity and BeamNG.tech), highlighting their advantages in safety, controllability, repeatability, and scalability, as well as their limitations related to the reality gap. Additionally, we compared two road representation formats central to our interoperability investigation (Chapter 5): OpenDRIVE, an expressive industry-standard format, and Catmull-Rom splines, a simple yet useful format in an academic-based simulator like Udacity. Finally, we introduced key technical tools and concepts used throughout the thesis, including DTW for sequence matching and agglomerative hierarchical clustering that underpin our test reduction and prioritization framework (Chapter 6).

The concepts and techniques presented in this chapter form a multidisciplinary foundation spanning software testing theory, DL systems engineering, autonomous vehicle architecture, and computational geometry. While classical software testing principles provide a useful starting point, the characteristics of DL systems, including continuous input spaces, data-driven behavior, and emergent properties, require adapted or novel approaches. Similarly, while simulation-based testing offers practical benefits for ADS validation, the proliferation of incompatible formats and the explosion of test scenario datasets create new challenges that cannot be addressed through traditional regression testing alone. The technical tools we introduced (DTW, road representation analysis) collectively provide the machinery necessary to reason about geometric and behavioral similarity in continuous ADS domains, distinguishing our approach from code-coverage or change-impact methods developed for discrete, explicitly-coded systems. With these foundations in place, we now review existing work in software testing, ADS validation, and test reduction to position our contributions within the broader research landscape.

Chapter 3

From DL Testing to ADAS Regression Testing

This chapter reviews existing literature across software testing, DL systems, and ADS, focusing specifically on testing practices, simulation environments, benchmarks, and advanced strategies for test selection and prioritization. It establishes a foundation and highlights gaps that the following chapters directly address.

3.1 Testing Practices in Open-Source Software Projects

Understanding testing practices in open-source software (OSS) projects provides an important baseline for assessing the maturity of software testing across ecosystems. Large-scale empirical studies of OSS projects have investigated how testing is adopted, automated, and evolved across different programming languages, domains, and project characteristics [135, 36, 136]. These findings offer general insights into how testing practices scale with project size, community involvement, and tooling support, independently of any specific application domain.

3.1.1 General OSS Testing Practices

Several empirical studies have examined testing practices in general OSS projects across multiple languages and ecosystems. Kochhar et al. [135] conducted one of the earliest large-scale investigations, analyzing over 20,000 GitHub repositories. They found that 61.65% of projects included test cases, with substantial variation across languages: C++ (78.8%), C (75.3%), and PHP (71.2%) showed higher test adoption than Python (37.5%). Their study also revealed strong correlations between testing adoption and project attributes such as team size, number of contributors, and issue activity, suggesting that testing is closely tied to project maturity and community engagement.

Islam et al. [36] revisited this line of work by focusing on Java projects from 2012 to 2021, demonstrating that test adoption increased significantly over time—from 67% to 81%—along with growth in test suite size. These results indicate a maturing testing culture in OSS, supported by improved tooling and CI/CD infrastructures.

Silva et al. [136] studied the adoption of testing tools across Go, JavaScript, PHP, and Python projects. While Python projects showed lower test adoption rates compared to other languages, they achieved relatively high average code coverage when tests were present. However, coverage practices varied widely, with many projects lacking any coverage reporting at all.

Overall, these studies characterize the state of OSS testing in terms of adoption, automation, and coverage, without focusing on any specific application domain.

3.1.2 Implications for Deep Learning Projects

The findings from general OSS studies are particularly relevant for DL systems, which are predominantly developed within open-source ecosystems and heavily rely on open-source libraries such as TensorFlow, PyTorch, and Keras. Moreover, many DL applications themselves are released as open-source projects, especially within the Python ecosystem, where DL development is concentrated.

As a result, understanding baseline OSS testing practices provides essential context for assessing the maturity of testing in DL projects. At the same time, general OSS studies do not explicitly account for the distinctive characteristics of DL systems, such as data-driven behavior, learned decision-making, and the absence of explicit functional specifications [137, 138]. This raises an open question: whether DL practitioners follow testing practices similar to those observed in traditional OSS projects, or whether DL projects exhibit systematic gaps, particularly in model-level and non-functional testing, that may be problematic for safety-critical domains such as ADS. This observation directly motivates our first research question (RQ_1).

3.1.3 Testing Practices in ML/DL Projects

Machine learning (ML) and deep learning (DL) systems introduce qualitatively new testing challenges compared to traditional software. As discussed in Section 2.2, DL systems are fundamentally data-driven rather than rule-based: their behavior emerges from learned parameters optimized on training data rather than from explicitly coded logic. Consequently, test oracles are often unclear, exhaustive testing is infeasible due to continuous input spaces, and traditional coverage metrics (e.g., statement or branch coverage) fail to capture behavioral adequacy [139, 140, 12].

While numerous surveys propose novel ML/DL testing techniques, relatively few studies empirically investigate how DL systems are tested in practice within open-source development. Santos [34] analyzed Python-based ML projects on GitHub and found that 65.19% employed at least one testing support library. However, their study did not distinguish between tests targeting ML models and those testing surrounding infrastructure, leaving open questions about model-specific testing adoption.

Vogelsang and Borg [141] highlighted fundamental challenges in requirements engineering and testing for ML systems, noting practitioners' reliance on offline accuracy metrics rather than comprehensive testing strategies. Serban et al. [142] further identified a gap between the perceived importance of trustworthiness practices (e.g., fairness, robustness)

and their actual adoption in ML projects. Similarly, Amershi et al. [143] reported that, despite recognizing the importance of model testing and monitoring, many teams struggle to operationalize these practices due to pipeline complexity and inadequate test adequacy criteria.

Building on these findings, our work (Chapter 4) presents a large-scale empirical study of 300 open-source Python DL projects. Unlike prior studies, we explicitly distinguish between general software tests and model-specific tests, quantify the adoption of automation and coverage practices, and identify maturity gaps that motivate the need for improved testing strategies in safety-critical DL applications.

3.2 Testing and Validation of ADS

ADS require thorough validation due to their complex modular architecture and safety-critical nature [144, 66]. Historically, ADS testing relied on both real-world experiments and extensive simulation in order to expose the system to diverse and challenging scenarios [144, 145, 146]. However, real-world testing alone is infeasible for several reasons: (1) safety risks: testing unverified ADS on public roads poses dangers to passengers, pedestrians, and property; (2) cost and time: physical testing requires vehicles, test drivers, instrumentation, and access to diverse environments; (3) limited scenario coverage: real-world testing cannot systematically expose ADS to rare, high-risk edge cases or reproduce failures consistently; and (4) reproducibility: real-world conditions are difficult to recreate precisely, hindering debugging and regression testing [27, 16].

As a result, simulation-based testing has become the dominant validation approach, offering a scalable, reproducible, and safe alternative that enables exposure to diverse scenarios without physical risks [147, 145]. However, this dominance introduces new challenges: scenario benchmarks have proliferated (SensoDat, DeepScenario, nuScenes, Waymo, etc.), each coupled to specific simulators and formats, creating fragmentation and limiting reusability; test suites have grown explosively as automated scenario generation techniques improve, making exhaustive execution prohibitively expensive; and there is a pressing need for principled methods to select, prioritize, and manage large-scale test campaigns across multiple platforms and ADAS architectures [145, 44].

Riccio et al. stress that ADS behavior evolves as a result of continuous interaction with road structures, surrounding objects, vehicle dynamics, environmental conditions, and control strategies, making deterministic testing infeasible [27]. Simulation-based testing has thus become the cornerstone of ADS validation, enabling systematic exposure to a wide range of scenarios without physical risks or costs [97, 148, 66]. In this section, we review ADS-specific test generation methods, benchmarks, and interoperability efforts that collectively define the current landscape and motivate the contributions of this thesis.

3.2.1 Benchmarks for ADS Testing

High-quality benchmarks are essential for standardized ADS evaluation and regression testing [146]. Benchmarks serve as reusable collections of test scenarios with known outcomes, enabling researchers and practitioners to: (1) compare different ADAS models fairly on a common set of cases, (2) detect regressions when ADS components are updated (e.g., when perception models are retrained), and (3) accumulate knowledge about which scenarios are most valuable for fault detection [145]. However, as test suites grow through automated scenario generation, managing and executing entire benchmarks becomes computationally expensive—the full SensoDat dataset requires over 100 hours of simulation per ADAS model [149]. This scalability challenge motivates the need for principled test selection and prioritization methods that preserve fault-detection capability while reducing execution cost, forming the basis of RQ_2 in Section 1.3.3. The following benchmarks align most closely with the objectives of Chapters 5 and 6.

3.2.1.1 SensoDat

Birchler et al. [40] introduced SensoDat, a large-scale benchmark comprising 32,580 road scenarios. In this thesis, a *scenario* refers to a single executable road specification (a sequence of geometric primitives, lane configuration, and initial vehicle state) that can be simulated in a driving simulator to observe ADAS behavior and determine pass/fail outcome. A *campaign* is a collection of scenarios generated by a specific test generation tool or algorithm under particular parameterization; SensoDat comprises three campaigns created by distinct generators.

The SensoDat scenarios were generated using test generator tools including Frenetic [150], Ambiegen [151], and Frenetic_v [152] (an extension of Frenetic). The dataset is organized into three scenario campaigns: *Ambiegen* (13 sub-campaigns), *Frenetic* (13 sub-campaigns), and *Frenetic_v* (10 sub-campaigns), each designed to target distinct road complexity profiles and geometric diversity. Each scenario is defined in OpenDRIVE format and was originally executed in the BeamNG.tech simulator using BeamNG's AI driver (a PID-based controller with global road knowledge).

SensoDat provides pass/fail labels based on whether the vehicle remained within lane boundaries (OOB criterion) and includes sensor data (camera images, LiDAR point clouds, vehicle telemetry) collected during execution [40]. While SensoDat is valuable for its scale and diversity, our investigation (Chapter 5) reveals a critical limitation: when converted to Catmull-Rom splines and re-executed with an independent DL-based ADAS model (DAVE-2) in the Udacity simulator, pass/fail ratios diverged by 25-50% compared to the original results, exposing tight coupling between benchmarks and specific ADAS architectures. This finding, that benchmark effectiveness depends not only on scenario

geometry but also on the ADAS model with which they were originally tested, motivates our subsequent research: (1) Chapter 5 investigates technical interoperability through OpenCat, enabling cross-platform reuse while quantifying the semantic generalization gap, and (2) Chapter 6 develops prioritization methods that identify fault-revealing scenarios with higher cross-model generalization, mitigating the architecture-dependent brittleness of benchmarks designed for single models.

The SensoDat benchmark is central to this thesis for three reasons: (1) it provides a large-scale, publicly available benchmark suitable for evaluating test reduction and prioritization methods (Chapter 6), and (2) its OpenDRIVE encoding and tight coupling to BeamNG.tech motivate our investigation of cross-platform interoperability in Chapter 5, and (3) the observed divergence in test outcomes across platforms and models highlights the need for behavior-aware prioritization that generalizes across ADAS architectures (Chapter 6).

3.2.1.2 CommonRoad: Multi-Format Scenario Benchmark

CommonRoad provides a large-scale motion planning benchmark suite featuring highway and urban environments with various traffic participants, including passenger cars, buses, and pedestrians. Scenarios are sourced from real traffic data (e.g., NGSIM dataset), simulations, and hand-crafted designs. Crucially, the framework also offers converters for multiple road representation formats, Lanelet/Lanelet2, OpenDRIVE, OpenStreetMap (OSM), and SUMO, underscoring its emphasis on cross-format interoperability [153, 154].

This emphasis directly aligns with the goals of our OpenCat converter. Scenario benchmarks represent significant intellectual and computational investment; enabling their reuse across different simulators, test platforms, and development environments multiplies their value and utility. Without interoperability, each new simulator or development environment requires either recreating scenarios from scratch or accepting vendor lock-in to a specific platform [145]. While CommonRoad's converters address interoperability by supporting multiple formats, they focus primarily on *route planning and motion planning* tasks, where the key concerns are lane topology, connectivity, and trajectory feasibility. Their OpenDRIVE-to-Lanelet converter prioritizes semantic road structure (lane connections, lane types) over precise geometric fidelity, which is appropriate for planning but insufficient for *control-focused testing* (e.g., lane-keeping ADAS) [153].

In contrast, our OpenCat converter uniquely targets *Catmull-Rom splines*, a lightweight, mathematically precise road representation suited for control testing in academic simulators like Udacity. This choice is critical for several reasons as described in 2.4.1.2. By enabling execution of industrial-strength benchmarks (SensoDat, OpenDRIVE-encoded scenarios) in lightweight academic simulators, OpenCat fills a gap between the expressiveness of

industry standards and the accessibility of research platforms, directly supporting the interoperability goals of RQ_2 [149].

3.2.1.3 DeepScenario

Lu et al. [41] introduced DeepScenario, a dataset of 80,000 driving scenarios collected from the CARLA simulator [29]. Scenarios include urban intersections, highway merges, and complex multi-agent interactions (vehicles, pedestrians, cyclists). DeepScenario emphasizes scenario diversity, covering a wide range of operational design domains (ODDs) [41].

However, DeepScenario illustrates a fundamental interoperability challenge: its scenarios are tightly coupled to CARLA's specific map layouts, town configurations, and scenario encoding format, limiting portability to other simulators [41]. Researchers wishing to evaluate their own ADAS models on DeepScenario scenarios must either use CARLA (accepting vendor lock-in and CARLA-specific physics/sensor models) or manually reconstruct scenarios in alternative simulators (a costly, error-prone process). This coupling directly exemplifies the interoperability barrier addressed by RQ_2 (Section 1.3.2). Solutions like OpenCat aim to decouple scenarios from simulators by converting between standard formats (OpenDRIVE to Catmull-Rom splines), enabling the same scenario to be executed and evaluated across diverse platforms without manual reconstruction [149].

Additionally, the original DeepScenario benchmark was evaluated using CARLA's built-in autopilot (a rule-based controller), raising questions about scenario generalizability to learning-based ADAS architectures [41]. This observation aligns with findings from Chapter 5: test outcomes can diverge significantly when executed with different ADAS models (e.g., PID-based vs. end-to-end learning), even when scenarios are technically portable across platforms. This motivates not only technical interoperability (format conversion) but also semantic interoperability (architecture-aware scenario evaluation), further motivating the behavior-aware prioritization methods in Chapter 6 that identify scenarios generalizable across diverse ADAS architectures.

3.2.1.4 SCTrans

Dai et al. [42] proposed SCTrans, which constructs a large public scenario dataset by mining real-world driving logs from public datasets (e.g., nuScenes [155], Waymo Open Dataset [156]). SCTrans translates the extracted scenarios into the standardized OpenSCENARIO format [157], enabling replay in various simulators. The dataset includes 50,000+ risky scenarios harvested from real-world driving data, improving realism beyond synthetically generated benchmarks [42]. Nonetheless, SCTrans encounters several challenges in scenario abstraction, including noisy sensor data in real-world logs and incomplete information about occluded objects, thus complicating accurate scenario re-

construction. Furthermore, similar to SensoDat and DeepScenario, SCTrans scenarios are format-dependent (OpenSCENARIO), which require conversion tools to enable their utility in simulators that do not support this format [42].

While benchmarks are valuable resources, their utility for regression testing is limited by two interconnected problems that directly motivate the contributions of this thesis:

Problem-1 (Format Compatibility): Benchmarks are often defined in simulator-specific formats (e.g., OpenDRIVE for BeamNG, OpenSCENARIO for CARLA) that are not natively supported by alternative simulators, limiting their direct reuse across platforms. Conversion tools (Section 3.4.2) are necessary to bridge this gap, but remain scarce and often incomplete. This problem motivates our RQ_2 addressed in Chapter 5, where we develop OpenCat, the first converter specifically targeting OpenDRIVE-to-Catmull-Rom transformation, enabling SensoDat scenarios to be executed in the Udacity simulator. OpenCat demonstrates that technical format conversion is feasible and can unlock access to large industrial benchmarks for academic research environments [149].

Problem-2 (Model-Specific Brittleness): Even when scenarios are technically portable across platforms, test outcomes can diverge dramatically when executed with different ADAS architectures. Benchmarks with pass/fail annotations based on a specific ADAS model (e.g., BeamNG's PID-based AI driver) often fail to generalize to architecturally distinct models (e.g., end-to-end learning networks). This problem is empirically demonstrated in Chapter 5: When SensoDat scenarios are converted and re-executed using DAVE-2 (a CNN-based end-to-end model) in the Udacity simulator, pass/fail ratios shift dramatically, from the original 61% passing rate to 98%, revealing that benchmark effectiveness depends not solely on scenario geometry but critically on the specific ADAS architecture for which it was designed [149]. This 25–50% outcome divergence exposes a fundamental interoperability limitation that format conversion alone cannot solve.

Benchmarks typically define scenario specifications using formats and semantics tailored to the environment they were created for, consequently reducing their effectiveness and generalizability in alternative contexts. This recurring limitation (reliance on specific ADAS models and simulators) hinders their utility for cross-platform and heterogeneous model testing [158, 146]. Addressing both problems together (format conversion + behavior-aware prioritization) enables a pathway toward more portable, generalizable, and reusable ADS benchmarks, which is critical as the field moves toward standardized validation frameworks and regulatory compliance requirements [96].

3.3 ADAS Regression Testing

In regression testing, test cases are attempted to minimize, select, or prioritize to reduce the execution cost without compromising fault-detection capability. Yoo and Harman [73] presented a comprehensive review of test minimization, selection, and prioritization techniques for traditional software, including coverage-based, history-driven, and requirement-based techniques. They concluded that it is not possible to identify a single best strategy that dominates across all contexts; hence, effective prioritization requires adapting to software and domain characteristics [73]. Later studies by Khatibsyarbini et al. [159] and Pan et al. [160] stress the importance of risk, execution history, and data-driven learning for effective prioritization.

However, classical regression techniques are based on discrete program logic and static coverage criteria, which cannot address ADS systems with a continuous, context-driven, and safety-critical behavior [161, 44]. In autonomous driving, behavior emerges from continuous interaction between the learned model and the driving environment. For example, two geometrically distinct road scenarios may execute identical code but produce drastically different vehicle trajectories and control outputs (e.g., safe lane-keeping vs. veering off-road). Conversely, minor code changes (e.g., hyperparameter updates, small weight adjustments during retraining) can produce major behavioral shifts on specific road geometries [12, 27]. Therefore, limiting the applicability of traditional regression techniques in the ADS context [161, 44].

Recently, research in ADAS regression testing has shifted towards the search-based, learning-based, and scenario-driven methods that explicitly integrate route geometry, multi-agent interactions, and vehicle behavioral responses. Three methodological trends are particularly relevant: *segmentation* of continuous driving sequences into semantically homogeneous subsections [162, 145, 148], *clustering* to detect and remove redundant scenarios while preserving diversity [163, 164, 148, 145], and *multi-criteria prioritization* that considers geometric complexity, behavioral risk, and historical fault information [32, 165]. While these trends demonstrate the necessity of behavioral and scenario-based approaches, existing methods often employ either geometry-only or behavior-only metrics in isolation. Our behavior-aware prioritization framework (Chapter 6) advances this line of research by integrating all three trends into a unified pipeline: (1) we segment roads via curvature analysis to create homogeneous geometric units, (2) we employ DTW-based clustering to group geometrically and behaviorally similar segments while preserving diversity, and (3) we apply multi-criteria prioritization, combining geometric complexity, dynamic driving features (speed variability, steering variability, cross-track error, yaw rate), and historical failures [134].

3.3.1 Search-Based Approaches

Birchler et al. [32] proposed SDC-Prioritizer, which uses meta-heuristic search (genetic algorithms, simulated annealing) combined with static road features (curvature, length, number of turns) to prioritize test cases for self-driving cars. Their evaluation in the BeamNG.tech simulator demonstrated that prioritization based on multiple geometric and topological features improves early fault detection compared to random and greedy-based test case ordering. However, they did not integrate dynamic ADAS behavior or historical failure data, both of which we incorporate in Chapter 6 [32].

3.3.2 Scenario-Based Approaches

In recent work, Amini et al [166] propose ITS4SDC, a bidirectional LSTM-based framework for classifying road scenarios as safe or unsafe to support test selection for lane-keeping systems in simulation. Using Frenetic-generated roads, geometric sequence features, and BeamNG-based pass/fail labels, the model prioritizes scenarios likely to trigger failures. ITS4SDC outperforms the ML-based SDC-Scissor baseline in accuracy and precision, demonstrating the value of sequence-aware deep learning for reducing redundant simulation runs, although performance degrades on smaller or distributionally different datasets, indicating limited generalization [166]. However, ITS4SDC does not enforce behavioral or geometric diversity, nor does it incorporate historical failure information during scenario selection, limiting its ability to ensure comprehensive and coverage-preserving testing for ADAS.

Similarly, Deng et al. [49] proposed STRaP, a scenario-based test reduction and prioritization framework that encodes driving recordings into semantic feature vectors. STRaP aligns and vectorizes recordings, segments them into unique scenarios based on static features (road type, weather) and dynamic features (relative velocities, distances to obstacles). to remove redundant tests, and prioritize the remaining tests using semantic coverage and scene rarity. Evaluated on the nuScenes dataset [155], the method reduces the test suite by over 34% while preserving nearly all fault-detection capability.

However, the approach focuses on perception-heavy urban scenarios (intersections, pedestrians) rather than control-focused lane-keeping scenarios. Additionally, their similarity metric relies on manually engineered features, which may not capture all relevant scenario properties [49]. We generalize Deng's segmentation approach (Chapter 6) to lane-keeping and replace manual feature engineering with automatic curvature extraction and DTW-based sequence matching, improving scalability and adaptability.

3.3.3 Clustering-Based Approaches

Kerber et al. [163] introduced a data-driven method for clustering highway driving scenarios using the highD dataset. Scenarios are extracted via a maneuver-based spatiotemporal feature filtering (i.e., vehicle trajectories, relative positions) approach. A custom distance metric combining Euclidean distance on feature vectors with temporal alignment via DTW measures similarity in surrounding-vehicle configurations around the ego vehicle, enabling hierarchical agglomerative clustering of structurally similar interactions, such as lane changes. By selecting representatives from each cluster, they reduced scenario count by 60% while maintaining coverage of behavioral diversity [163].

We extend Kerber's approach (Chapter 6) by integrating the dynamic ADAS behavior data (e.g., speed and steering variability) into the clustering process, enabling more accurate cluster formation through the combined use of geometric and behavioral features. Our evaluation demonstrates that this hybrid clustering produces more diverse and informative test selections. Moreover, while Kerber's highway-focused method shows performance degradation when applied to lane-keeping scenarios, our approach maintains robust performance across both DAVE-2 and Chauffeur models, indicating improved generalizability and consistency.

Likewise, Bernhard et al. [164] proposed a multi-stage trajectory clustering method that combines Gaussian Mixture Models (GMM) and hierarchical clustering to group vehicle interactions based on spatial and directional behavior, enabling effective scenario diversity management. Their analysis indicates that fault-detection can still be guaranteed for the reduced clustered scenarios, revealing a trade-off between redundancy reduction and test performance [164, 49].

While their method focuses on clustering multi-agent interactions, our work (Chapter 6) targets single-vehicle lane-keeping control and extends clustering by jointly integrating geometric road features and dynamic ADAS behavior. Unlike their approach, which is limited to redundancy reduction, we further introduce a multi-criteria prioritization strategy that ranks scenarios based on difficulty and failure relevance. This enables not only effective scenario reduction but also significantly improves early fault detection, enhancing the overall efficiency and robustness of the testing process.

Our work (Chapter 6) extends these approaches by integrating geometric road features (curvature profiles) with dynamic ADAS behavior (speed, steering, cross-track error, yaw rate), and historical failures, enabling redundancy elimination and multi-criteria prioritization. In contrast to prior work, which either prioritizes without ensuring diversity or clusters without ranking, we provide a unified, behavior-grounded pipeline tailored to single-vehicle lane-keeping control for ADAS regression testing.

3.4 Interoperability and Format Conversion in ADS Testing

Scenario benchmarks represent significant intellectual and computational investment; enabling their reuse across different simulators, test platforms, and development environments multiplies their value and utility. *Interoperability:* the ability to reuse test artifacts across simulators and ADAS models is essential for scalable testing but remains under-explored. Without interoperability, each new simulator or development environment requires either recreating scenarios from scratch or accepting vendor lock-in to a specific platform [145]. This fragmentation hinders independent validation: researchers using different simulators cannot directly compare ADAS performance on the same scenarios, and proprietary benchmarks become inaccessible once their original platform becomes obsolete or unsupported. Interoperability also enables cross-platform robustness evaluation by executing the same scenario on multiple simulators. We can distinguish between ADAS weaknesses and simulator artifacts, informing the design of more robust controllers [31]. This motivation directly underpins RQ_2 in (Section 1.3.2)

Despite decades of standardization efforts, adoption remains inconsistent across the ADS ecosystem [167]. While industrial simulators, such as CARLA [29], VTD [168], and BeamNG.tech [30], support OpenDRIVE natively, enabling interchange of complex road networks. However, several academic research-oriented and lightweight simulators (e.g., Udacity [47], TORCS [169], custom in-house tools) employ proprietary or simplified road representations such as waypoint sequences, polynomial curves, or custom geometric primitives [167, 145]. This fragmentation creates concrete barriers for benchmark portability and cross-platform validation [167]. Benchmarks published in simulator-specific formats, such as OpenDRIVE (e.g., SensoDat), can only be executed in simulators that natively support these formats, thereby restricting their usability, particularly for academic researchers who lack access to or familiarity with industrial simulation platforms. As a result, researchers seeking to use industrial benchmarks in academic simulators must manually recreate road geometries, a time-consuming and error-prone process that introduces geometric inaccuracies and significantly limits benchmark reuse. Consequently, large-scale benchmarks such as SensoDat remain largely inaccessible to researchers operating on alternative platforms, hindering independent evaluation and compromising reproducibility. These barriers directly motivated our OpenCat converter, addressing the gap between standards adoption and practical accessibility for academic research [149].

To address this interoperability gap between industry standards and academic simulators, we develop OpenCat (Chapter 5), the first converter specifically designed to transform OpenDRIVE road representations into Catmull–Rom splines, thereby enabling the execution of industry-standard benchmarks in lightweight research simulators such as Udacity. OpenCat achieves 100% conversion accuracy ($R^2 = 1$) across all 32,580 SensoDat scenar-

ios, preserving geometric fidelity, including curvature profiles and elevation data, while remaining computationally efficient. By bridging OpenDRIVE (an industry standard) and Catmull–Rom splines (commonly used in academic simulators), OpenCat directly addresses RQ_2 by enabling cross-platform scenario reuse and independent ADAS evaluation, which are foundational for scalable and reproducible ADS testing [149].

3.4.1 Interoperability

The proliferation of ADS simulators (CARLA [29], BeamNG.tech [30], and LGSVL [170]) raises the fundamental questions about the portability of scenarios and consistency of results across platforms [171, 31, 172].

Cross-simulator discrepancies and their causes: Borg et al. [31] investigated cross-simulator transferability of ADS models and introduced the phrase *digital twins are not monozygotic* to characterize systematic discrepancies in test outcomes that arise from simulator-specific design choices rather than random noise. These systematic discrepancies manifest as differences in pass/fail ratios, failure locations (e.g., which curve causes lane departure), and ADAS behavior patterns when identical scenarios are executed in different simulators [31]. By executing identical lane-keeping scenarios in BeamNG and Udacity, they revealed substantial differences in test outcomes despite geometrically equivalent roads, attributing these discrepancies to differences in physics engines (e.g., vehicle dynamics modeling, tire friction models), sensor noise models (e.g., camera blur, LiDAR measurement uncertainty), and rendering pipelines (e.g., lighting, texture quality affecting visual perception) [31].

Our study of the converted SensoDat dataset (Chapter 5) corroborates and extends these findings. When identical scenarios are executed in BeamNG.tech (original environment) and Udacity (via OpenCat conversion) using an independent ADAS model (DAVE-2), we observe significant pass/fail redistribution: SensoDat's original 61% pass rate shifts to 98% pass rate in Udacity, a 37 percentage-point divergence [149]. This gap cannot be attributed to conversion errors (geometric accuracy is $R^2 = 1$) but rather reveals that simulator physics, sensor models, and vehicle dynamics implementations produce fundamentally different driving experiences for the same road geometry.

Simulator reliability and non-determinism: Amini et al. [172] studied simulator fidelity and identified *flaky simulators*, platforms where re-executing the exact same scenario multiple times yields inconsistent results due to numerical instabilities, floating-point precision differences, or non-deterministic physics engines [172]. This simulator's non-determinism creates a *reproducibility problem* for regression testing: if the same test case produces different results across runs due to simulator artifacts rather than

genuine ADAS behavior changes, it becomes impossible to distinguish real regressions from simulator noise. This undermines the core value of regression testing, which relies on deterministic test outcomes to detect genuine faults introduced by code or model changes [172].

To address this, Amini et al. proposed robustness metrics such as *execution consistency*, the percentage of repeated test executions yielding identical outcomes, and recommended multi-simulator testing to mitigate platform-specific biases [172]. Their recommendation that researchers evaluate scenarios across multiple datasets and simulators reflects the understanding that single-platform validation is insufficient for confident ADAS assurance.

While Amini's work focuses on simulator reliability and mechanical reproducibility, our study in Chapter 5 reveals a deeper, model-dependent layer of interoperability: even with reliable, deterministic simulators, ADAS model architecture fundamentally affects test outcomes. PID-based controllers with global road knowledge (BeamNG's AI driver) perceive and respond to scenarios differently than end-to-end learning models (DAVE-2) that infer steering from visual input alone [149]. This architectural difference is not a simulator artifact but a genuine feature of how diverse ADAS systems interact with the same road geometry.

If benchmarks are tightly coupled to specific ADAS models, their value for independent evaluation and cross-platform validation is severely limited. Researchers testing alternative architectures (e.g., a new end-to-end learning approach, a modular perception-planning-control pipeline) cannot reliably reuse existing benchmarks, as test outcomes will be architecture-dependent rather than reflecting universal scenario properties [149]. Model-independent benchmarks, defined by observable safety properties (e.g., vehicle must not exit lane) rather than model-specific metrics—enable fair evaluation across diverse ADAS approaches and support the standardization goals emphasized in emerging regulations like UNECE GTR [96].

The results from these studies collectively demonstrate that technical interoperability (format conversion, cross-platform execution) does not guarantee *semantic equivalence* in scenario criticality, the notion that a scenario's safety-critical properties remain consistent across simulators and models [149, 31]. In other words, even when scenarios are geometrically identical and technically portable, their ability to reveal failures is strongly dependent on the simulator's physics and sensor models, as well as the specific ADAS architecture being tested [149]. A scenario that challenges a PID controller (e.g., a sharp curve requiring precise steering) may pose no difficulty for an end-to-end learning model trained on similar curves, or vice versa. This semantic gap means that criticality judgments (e.g., this scenario is high-priority because it fails 50% of test runs) may not transfer between simulators or models.

This finding directly motivates our behavior-aware prioritization framework. If scenario criticality is not universally transferable, test prioritization strategies must account for both geometric and behavioral patterns that correlate with failures across multiple ADAS models, rather than relying on model-specific failure history alone. By integrating dynamic driving behavior metrics (speed variability, steering variability, cross-track error) with geometric features, our framework identifies scenarios that challenge diverse ADAS architectures, enabling prioritization that generalizes better across models and simulators [134]. This hybrid approach helps bridge the semantic interoperability gap exposed by cross-simulator and cross-model studies.

3.4.2 Format Conversion Tools

Scenarios are often defined in simulator-specific formats (e.g., OpenDRIVE for BeamNG, OpenSCENARIO for CARLA) that are not natively supported by alternative simulators, limiting their direct reuse across platforms. Several studies have been conducted in an attempt to convert different road representations to address the format incompatibility issue. Lin et al. [153] present a tool for automatic conversion from OpenSCENARIO to CommonRoad [173], a format that is used in motion planning research. First, the tool parses OpenSCENARIO XML, extracts road geometries and vehicle trajectories, and then reconstructs equivalent CommonRoad scenarios. However, it is limited to 2D road networks (i.e., with no elevation) and does not support all OpenSCENARIO features (e.g., conditional triggers, stochastic events) [153].

Queiroz et al. [174] developed an OpenDRIVE to Lanelet2 converter [175], Lanelet2's lane-level topology format for autonomous navigation. Their tool prioritizes lane-connectivity information, sacrificing geometric fidelity in favor of routing efficiency. Preserving geometric fidelity is essential for controlling ADS, even small geometric errors compound across long driving scenarios, potentially introducing artificial failures or masking real ones. While effective for planning tasks, this conversion is insufficient for control-focused testing, where precise geometric accuracy is essential [174].

Bock et al. [176] introduce a textual domain-specific language (DSL) and the scenario-accompanied, text-based, iterative Evaluation of automated driving Functions (stiEF) methodology to support structured, multilingual scenario specification for automated driving function (ADF) development. The DSL enables unambiguous scenario descriptions across multiple abstraction levels and defines scenarios using high-level primitives (e.g., roads, vehicles, sensors) and then compiled into simulator-native formats such as OpenDRIVE and OpenSCENARIO.

However, the approach still requires maintaining separate format-specific compilers for each target simulator, limiting portability. Moreover, while stiEF supports abstraction levels

in textual modeling, it does not guarantee consistency across the artifacts generated for different simulators. As a result, semantic discrepancies such as variations in physics, sensor noise, or actuator models remain unresolved, constraining its usefulness for cross-platform testing and validation [176]. While Bock focuses on representation abstraction, we expose (Chapter 5) that semantic mismatches persist even with perfect geometric conversion, revealing a deeper interoperability challenge that representation abstraction alone cannot solve.

Our work (Chapter 5) contributes to this line of research by developing OpenCat, a tool for converting road representations between formats (OpenDRIVE and Catmull-Rom splines), enabling cross-platform testing. Unlike prior converters that target perception or planning tasks, OpenCat prioritizes geometric fidelity for control testing, achieving 100% conversion accuracy (measured by $R^2 = 1$ between original and converted curvature profiles). However, our findings align with Borg et al. [31]: even with perfect geometric conversion, test outcomes differ substantially due to model-specific brittleness, not merely simulator differences.

We contribute to this area of research (Chapter 5) by implementing OpenCat, a tool that can transform road representations between different formats: OpenDRIVE (the industry standard) and Catmull-Rom splines on top of it, in order to allow testing across platforms. Compared to previous converters, which focus on perception or planning purposes, OpenCat favors geometric accuracy for control testing and attains 100% conversion accuracy. But, our results are consistent with Borg et al. [31]; even with ideal geometric conversion, test results are widely different because of model-specific brittleness, not merely the simulator differences.

3.5 Identified Research Gaps

The literature review reveals several gaps that motivate and situate the contributions provided in this thesis.

3.5.1 Limited Empirical Understanding of DL Testing Practices

While testing practices in general open-source software have been extensively studied [135, 36, 136, 177], only a few studies specifically target deep learning projects [34, 141]. Santos [34] provides preliminary insights but does not distinguish between tests for ML models versus surrounding infrastructure (data pipelines, training loops, data validation). Consequently, critical questions related to test suite evolution, test automation integration, coverage practices, and model-specific testing remain unanswered. This gap directly motivates our study in Chapter 4, which presents the first large-scale empirical study of testing practices in 300 Python DL projects, quantifying the adoption of testing methodologies,

identifying maturity gaps in model-specific and non-functional testing, and establishing an empirical baseline for assessing the readiness of DL systems for safety-critical deployment.

3.5.2 Benchmark Portability and Model-Specific Brittleness

Existing ADS benchmarks [40, 41, 42] provide valuable scenario collections but suffer from two fundamental limitations. (1) *Format Compatibility*: Benchmarks are encoded in simulator-specific formats (e.g., OpenDRIVE for BeamNG, OpenSCENARIO for CARLA), preventing direct reuse in alternative simulators, and (2) *Model-Specific Brittleness*: Benchmarks are implicitly coupled to the ADAS models with which they were created, causing test outcomes to diverge dramatically when scenarios are re-executed with architecturally different models. This gap directly motivates our investigation in Chapter 5, which introduces OpenCat, the first converter enabling cross-platform scenario reuse by transforming OpenDRIVE roads into Catmull-Rom splines with 100% geometric accuracy, while empirically demonstrating that technical format conversion alone does not resolve the deeper problem of model-dependent benchmark behavior, revealing the need for architecture-agnostic testing approaches.

3.5.3 Scalable Regression Testing for DL-Based ADAS

Test suite growth (identified in Section 3.5.1) necessitates principled test reduction and prioritization strategies. However, classical regression testing techniques such as those based on code coverage [79, 83, 73] are fundamentally inadequate for DL-based ADAS, where behavior emerges from learned parameters rather than explicitly coded logic. Recent ADS-specific prioritization methods [49, 178, 179, 32] represent progress toward scenario-aware approaches but face significant limitations: they often rely on single-modality features (geometric properties alone) without integrating dynamic driving behavior, lack validation across diverse ADAS architectures, and do not systematically address the cross-model generalization problem revealed in Section 3.5.2. This gap directly motivates our investigation in Chapter 6, which develops a behavior-aware, coverage-guided prioritization framework that integrates geometric road features (curvature profiles) with dynamic driving metrics (speed variability, steering variability, cross-track error, yaw rate), achieves 89% test suite reduction while retaining 79% of failures and improving early fault detection by up to 95%, and demonstrates meaningful cross-model generalization across diverse ADAS architectures.

Part II

Empirical Investigation and Core Contributions

Chapter 4

Understanding Testing Practices in Deep Learning Projects

4.1 Testing of Deep Learning Projects

As established in Chapter 2, DL systems exhibit fundamental characteristics including data dependency, non-determinism, continuous input spaces, and emergent behavior that distinguish them from traditional software and complicate conventional testing approaches. Chapter 3 reviewed the state of testing in general OSS and highlighted a critical gap: while empirical studies exist for traditional software ecosystems, there is limited understanding of how DL systems are actually tested in practice, particularly regarding model-specific validation and test suite evolution. This chapter directly addresses that gap through the first large-scale empirical study of testing practices in 300 Python DL projects, establishing an empirical foundation that motivates and informs the subsequent research on autonomous driving system testing in Chapters 5 and 6.

With the rise of DL systems in safety-critical areas such as ADS and medical diagnosis, there has been an increasing need for rigorous testing practices to mitigate faults and build trust [12]. Unlike traditional software, where behavior is encoded through deterministic logic, DL systems derive their behavior from learned parameters optimized during training on historical data [3, 180]. This data-driven approach brings its own challenges: models are susceptible to adversarial perturbations (small, crafted input modifications causing misclassification), lack of interpretability (difficulty understanding why a model made a specific decision), and can catastrophically fail on out-of-distribution (OOD) inputs [89, 59, 84].

For example, a lane-detection model trained on sunny highway driving may fail catastrophically when encountering rain-obscured lane markings or night-time driving with different lighting situations that fall outside the distribution of training data [181, 182]. More insidiously, adversarial perturbations such as small stickers placed on road signs or subtle texture changes on lane markings can cause traffic sign classifiers to misidentify signs or lane-detection models to drift out of lane, despite the perturbations being imperceptible to human observers [69]. These failure modes are qualitatively different from traditional software bugs and demand specialized validation strategies.

Although testing is fundamental to software quality, empirical evidence suggests significant immaturity and variability in testing practices across DL projects [12, 59]. Existing

studies have examined testing adoption in general OSS projects [135, 36, 136] and specific ecosystems such as Android applications [177], but there has been limited systematic investigation of testing practices specifically in deep learning projects. Moreover, prior work that touches on ML/DL testing (e.g., Santos [34]) does not distinguish between tests for the DL model itself versus tests for surrounding infrastructure (data pipelines, training loops, data validation utilities). Consequently, critical questions remain unanswered:

- What percentage of DL projects include test suites, and of those, how many specifically validate model behavior?
- Which testing frameworks and automation tools are adopted in DL projects, and are they integrated into continuous integration (CI/CD) workflows?
- Do DL projects measure and report code coverage, and if so, do coverage levels differ between general code and model-specific code?
- How do test suites evolve as DL projects mature? Do they exhibit unique growth patterns compared to traditional software?

This gap is particularly concerning because DL projects face unique validation challenges beyond traditional functional correctness: validating model accuracy, robustness to adversarial perturbations and distribution shifts, fairness across demographic groups, and interpretability of decisions [59, 84, 183]. Understanding current practice is essential for identifying maturity gaps and designing testing approaches tailored to DL characteristics, especially for safety-critical applications like ADS, where inadequate testing can result in loss of life.

This chapter presents the first major empirical contribution of this thesis: a large-scale investigation into testing practices within Python deep learning projects hosted on GitHub. Building on the gaps identified in Chapters 2 and 3, we systematically study 300 carefully *selected* (not randomly sampled) open-source DL projects spanning three popular frameworks: TensorFlow [184], PyTorch [185], and Keras [186]. We investigate the adoption of testing methodologies, automation frameworks (e.g., PyTest [187], UnitTest [188]), coverage practices, and test suite evolution patterns, with particular emphasis on distinguishing general infrastructure tests from *model-specific tests*, those designed to validate the behavior of trained neural networks.

Our findings reveal both promising trends and critical gaps. While test automation adoption is high (86.63%), only 55% of projects with test suites include model-specific tests, and non-functional testing (performance, security) is severely underrepresented. Moreover, test suites exhibit rapid growth (up to 10×), indicating that regression testing efficiency will become increasingly important as DL projects mature. These observations establish the empirical motivation for our subsequent work on ADS testing, including the

interoperability and prioritization research in Chapters 5 and 6.

The rapid test suite growth observed in this chapter directly motivates the need for efficient regression testing in ADAS contexts, including the focus on test prioritization and reduction methods. Additionally, the gap in model-specific testing highlights that even in open-source projects, systematic validation of learned behavior remains immature; a concern amplified in safety-critical domains like autonomous driving, where model failures can have catastrophic consequences. Our subsequent contributions directly address this maturity gap by developing tools and methods for cross-platform benchmark reuse (OpenCat, Chapter 5) and behavior-aware test prioritization (Chapter 6), enabling scalable, efficient, and architecture-aware testing of DL-based ADAS.

4.2 Study Design

We employ a mixed-methods approach that combines quantitative analysis of repository metadata (i.e., commit logs, file structures, workflow configurations) with qualitative examination of test code content. This systematic approach enables us to investigate the current testing practices in DL projects, particularly for model-specific testing and test evolution (RQ_1, Chapter 1).

4.2.1 Research Questions

To systematically investigate testing practices in Python deep learning projects and provide a comprehensive answer to RQ_1 (Chapter 1), we decomposed the overarching research question into four specific sub-questions (RQ_{1a}–RQ_{1d}), each examining a different dimension of how DL projects approach testing.

RQ_{1a} (Test Case Presence and Types): *Are Python DL GitHub projects tested? What type of tests are implemented?*

This sub-question investigates the breadth of testing adoption in DL projects and the diversity of testing approaches employed. We examine whether projects include test suites, distinguish between infrastructure tests (utilities, data pipelines, training orchestration) and model-specific tests (tests directly validating learned behavior such as inference correctness, robustness to distribution shifts, or adversarial inputs), and characterize the distribution of test types across functional categories (unit, integration, system) and non-functional categories (performance, security, smoke testing).

RQ_{1b} (Test Automation Adoption): *What is the adoption rate of test automation in Python DL GitHub projects?*

This sub-question quantifies the usage of automated testing tools and examines the integration of testing into continuous integration (CI) and continuous deployment (CD) pipelines. Test automation frameworks (PyTest, UnitTest, etc.) enable rapid, reproducible, and error-free test execution, forming the backbone of modern development practices. However, merely adopting an automation framework is insufficient if tests are not automatically triggered on every code change. We investigate which automation frameworks are adopted, how frequently, and whether projects integrate automated test execution into CI/CD workflows using platforms such as GitHub Actions, Travis CI [189], or Jenkins [58].

$\mathbf{RQ_{1c}}$ (Code Coverage Analysis): *Are test coverage criteria considered in Python DL GitHub projects?*

Test coverage metrics quantify the proportion of source code exercised during testing. While coverage is an imperfect proxy for test quality, it provides visibility into untested code paths and guides test effort allocation. This sub-question investigates the adoption of coverage tools (Codecov, Coveralls, Scrutinizer) and coverage levels achieved, with particular attention to whether projects measure coverage separately for model-specific code versus infrastructure code.

$\mathbf{RQ_{1d}}$ (Test Suite Evolution and Maintenance): *How do test suites evolve in Python DL GitHub projects?*

Measuring test suite size and modification rates over time provides insights into maintenance burden and growth patterns. We track additions, modifications, and deletions of test files across multiple releases, analyzing how test suites scale as projects accumulate features and mature. Rapid growth (e.g., $10\times$ expansion in 10 releases) signals that regression testing will become increasingly expensive, motivating the need for systematic test management strategies, directly relevant to the prioritization research in later chapters and motivating RQ_3 from Chapter 1.

4.2.2 Project Selection

Selecting representative DL projects to be thoroughly analyzed requires balancing breadth (covering diverse projects) with depth (ensuring projects are substantial and actively maintained). To achieve this balance and meaningfully address our research questions, we adopted a multi-stage selection process for identifying relevant Python-based DL projects on GitHub. We target Python as the primary language for our investigation due to its widespread use in DL application development [190].

Initial Search: We used the GitHub Search API [191] to retrieve projects with Python as our primary language for DL development [190], combined with one of three popular

DL libraries: TensorFlow [184], Keras [186], and PyTorch [185]. These frameworks are widely adopted and reflect the ongoing trends in deep learning development [192, 193, 194]. This resulted in 50,500 TensorFlow projects, 24,800 Keras projects, and 64,000 PyTorch projects. Since it is infeasible to analyze all 139,300 projects, we introduced rigorous project selection criteria as described below.

Project Selection Criteria: We applied exclusion criteria to ensure that the selected projects were non-trivial, actively maintained, and widely adopted. Using GitHub's advanced search, we filtered for repositories with at least five contributors (via the *"followers≥ 5"* filter) and those that demonstrated strong community adoption by requiring more than 100 stars and 100 forks (using the *"stars > 100"* and *"forks > 100"* filters, respectively). These metrics indicate the repository's impact and popularity because users typically star, watch, or fork repositories they find useful and interesting, thereby increasing their visibility within the GitHub community [135, 36].

Applying these filters narrowed the candidate set to 783 TensorFlow, 291 Keras, and 1,200 PyTorch projects. These filters are consistent with prior empirical studies of open-source projects [135, 36] and balance inclusivity (avoiding overly restrictive filters that exclude valuable projects) with quality (excluding dummy projects, tutorials, or abandoned repositories).

Manual Validation and Selection: To ensure projects represent genuine, actively maintained DL systems, we manually excluded projects falling into several categories:

- **Inactive repositories:** Projects with fewer than 100 commits, indicating minimal development activity. The 100-commit threshold was chosen to filter out toy projects, proof-of-concept experiments, and projects with trivial implementation scope, while retaining projects with meaningful engineering effort [135].
- **Recently abandoned projects:** Repositories with no commits in the preceding 12 months, indicating the project is no longer actively maintained. Projects abandoned for over a year are unlikely to reflect current testing practices and may have technical debt or deprecated dependencies.
- **Non-code repositories:** Tutorials, books, code example collections, and documentation-only repositories that do not represent functional DL systems or applications.

These exclusion criteria yielded a refined candidate pool. From this pool, we selected 100 projects per framework (300 total) using deterministic ranking by GitHub star count, rather than random sampling. Star count serves as a proxy for project impact, community adoption, and perceived quality [195]. By selecting the top 100 highest-starred projects per framework, we ensure our sample includes widely used, influential projects whose testing

Table 4.1: Distribution of stars, forks, and contributors per framework

Framework	Stars			Forks			Contributors		
	Max	Avg	Min	Max	Avg	Min	Max	Avg	Min
TensorFlow	115,000	121	8982	206,636	2066	103	2,183	16684	5
PyTorch	115,000	11602	103	83,000	4550	62	10,900	299	7
Keras	27,300	4369	169	27,300	106621	42	843	80	5

practices are likely to reflect community standards and best practices. This non-random selection strategy prioritizes coverage of the most visible and adopted projects, maximizing the relevance of our findings to the broader DL development community, though it may bias results toward projects with higher visibility (potential limitation discussed in Section 5.5).

Table 4.1 summarizes the distribution of stars, forks, and contributors among the selected 300 projects. The material used to conduct our study is publicly available at [196].

4.2.2.1 Data Extraction Methodology

For each selected project, we extracted data from three sources: repository structure, version control history, and documentation. Data extraction was performed between January 2024 and March 2024 using the GitHub REST API (v3).

RQ$_{\mathbf{1a}}$(Test Case Presence and Types): We determined test suite presence by checking for common naming conventions for test directories (*tests/*, *test/*) and test files (*test_ *.py*, **_ test.py*). We also searched for test-related configuration files (*pytest.ini*, *setup.cfg*, *tox.ini*), which indicate testing infrastructure setup. When initial structure analysis did not reveal tests, we performed targeted keyword searches using GitHub's search API for each repository. This multi-stage approach reduces false negatives, though we acknowledge that unconventional test organization (e.g., tests embedded in source code or stored outside the repository) may be missed.

Identifying Model-specific Tests: Since manually inspecting every test file across 300 projects is infeasible, we employed automated detection via keyword matching on file and directory names. However, we acknowledge that some keywords may be ambiguous: for example, *'test_ regression'* could refer to statistical regression testing or regression testing of a regression model, while *'test_ classification'* could refer to functional test classification or validation of a classification neural network. To mitigate this ambiguity, we used a combined approach:

1. **Framework-based signals:** If a test file imports DL frameworks (e.g., *import tensorflow*, *from torch import nn*) and contains keywords related to model behavior, we classify it as a model test.

2. **Keyword phrases:** We searched for keywords specifically associated with model validation: *'model'*, *'prediction'*, *'training'*, *'inference'*, *'layers'*, *'evaluation'*, and combinations such as *'model_test'*, *'test_prediction'*, *'test_training'*, *'test_inference'*, *'test_regression'*, *'test_classification'*.

Due to the ambiguity of some keywords and the limitations of automated detection, our model-specific test counts should be interpreted as lower-bound estimates: we are confident these projects have model tests, but we may have undercounted projects where model tests exist but use unconventional naming.

Determining the Types of Tests: Since tests serve different purposes, validating different aspects of a developed project, our investigation characterizes the distribution of test types. We distinguish between test levels (unit, integration, system) and test types (functional, non-functional), as defined in Section 2.1.2.

To identify test levels and types, we employed a multi-faceted approach: (1) searching for common naming conventions in test folders and files such as *'unit_test'*, *'integration_test'*, *'system_test'*, *'performance_test'*, *'smoke_test'*, and *'security_test'*; (2) keyword searching within test files for terms like *'unittest'*, *'integration'*, *'performance'*, *'load'*, *'stress'*, and *'concurrent'* that indicate test purpose; and (3) examining test code structure, for example, tests involving multiple components or end-to-end workflows are classified as integration or system tests, while performance tests are further identified by keywords like *'load'*, *'stress'*, and *'concurrent_user'* that measure system behavior under specific conditions. We acknowledge that this automated, keyword-based classification may misclassify tests with unconventional or ambiguous naming; consequently, reported test type distributions should be interpreted as approximations rather than exact counts.

$\mathbf{RQ_{1b}}$(Test Automation Adoption): Automation tools play an important role in test case execution. Manual testing is usually expensive, prone to human errors, and hardly reproducible. In contrast, automated test execution usually offers a more robust, reliable, and repeatable testing process, although the quality of the resulting process depends on the quality of the underlying test scripts and frameworks [197, 39].

In this sub-question, we investigate whether the GitHub test suites in our selected projects use any automation tools to execute test cases. To do this, we reviewed each repository's test files, codebase, and workflows to collect data about the automation tools. Although we selected Python as our primary language, our analysis revealed that several projects also incorporate other languages such as C++, C#, Java, JavaScript, and Node.js within their code bases. Consequently, we not only considered Python-specific test automation tools (e.g, PyTest, UnitTest, Jest) but also tools associated with these additional languages to get a better picture of the automation practices present in the selected repositories.

To identify the automation tools used in the test suites, we searched for directory patterns commonly used by developers when organizing their code bases. This includes folder names such as *'test/automation'*, *'tests/(language_ name)/automation'*, as well as file naming conventions like *'auto_ test*.py'* or *'auto_ test*.js'*, which suggests the presence of test automation frameworks.

We then searched for widely used testing frameworks for the relevant programming languages in our selected projects: *PyTest* [187], *UnitTest* [188], *JUnit* [198], *TestNG* [199], *Jest* [200], *Mocha* [201], *GTest* [202], *xUnit* [203] for *Python*, *Java*, *JavaScript*, *Node.js, C++*, and *C#*, respectively [204, 35]. This involved searching for keywords including pytest, unittest, junit, testNG, jest, mocha, gtest, tox, and xunit to identify the testing frameworks.

Additionally, we also reviewed the project's documentation and README file, seeking any indication of test automation frameworks. During our investigation, we discovered that out of the several projects reviewed, only a small subset of them, specifically 20 out of 232 projects (with test suites), provided additional information about their testing practices, supplementing our initial findings.

GitHub workflows are widely used to automate test execution [39]. In our analysis, we found that more than half, precisely 161 out of 300, of the selected projects contained workflow folders/files within their repositories. To extract additional information about testing practices, we carefully inspect the contents of the GitHub/workflows directory to identify any workflow files typically written in YAML format (i.e., ci.yml, travis.yml, and config.yml). These files serve as blueprints for defining the necessary steps to build, test, and deploy the codebase [39]. We specifically focused on identifying workflow files related to testing, as their presence indicates the use of an automated test suite. Apart from these workflow (yml) files, some workflows contain specific automated tasks, such as installing test requirements, debugging the CLI, running tests, and uploading coverage reports.

We also searched for Continuous Integration/Continuous Deployment (CI/CD) tools, including TravisCI, Jenkins, CircleCI, and GitHub Actions, which are often used to automate testing. Within each workflow, we reviewed steps involving explicit test commands or testing frameworks. Commands such as *'npm test'*, *'run tests'*, *'pytest'*, *'pytest-cov'*, *'run tox'*, *'mocha'*, or similar served as evidence of automated testing practices. These commands may be manually executed by developers or seamlessly integrated into continuous integration (CI) or continuous delivery (CD) pipelines to ensure consistent and automated test execution throughout the development lifecycle [205, 39].

RQ_{1c}(Code Coverage Analysis): Code coverage metrics measure the proportion of source code exercised by a test suite [206]. Common coverage metrics include line coverage

(percentage of executable lines executed), branch coverage (percentage of conditional branches exercised), statement coverage (percentage of individual statements executed), function coverage (percentage of functions or methods called), and condition coverage (percentage of boolean sub-expressions evaluated to both true and false) [207]. Code coverage metrics serve as valuable indicators of test suite comprehensiveness and can help identify code areas requiring additional testing attention, thereby reducing the risk of bugs escaping detection and impacting production systems [208]. In this investigation, we examine the extent to which Python DL projects consider and report test coverage criteria, focusing on evidence found in project documentation, configuration files, and commit messages.

To answer RQ_{1c}, we first examined project repositories to find the presence of test coverage reports. This includes examining commit messages, code comments, and discussions related to test completeness. We searched for comments that explicitly mention test coverage metrics (e.g., statement coverage, branch coverage), test coverage tools, or discussions about improving the thoroughness of the test suite.

Manually setting up and running coverage tools across numerous projects locally is both time-consuming and prone to errors. To streamline this process, we prioritized publicly available coverage information found in the project documentation. We began by thoroughly examining project READMEs. Ideally, READMEs provide labels denoting the employed coverage tools, offering easy access to comprehensive reports.

To strengthen our investigation and ensure we captured all potential test coverage practices, we also examined documentation beyond the README files for mentions of code coverage tools. We inspected the '.github/workflows' folder for files specifically associated with code coverage tools. We searched for filenames commonly associated with coverage tools, such as *'.codecov.yml'*, *'.coveragerc'*, *'.coveralls.yml'*, and *'coverage.py'*. For the projects with identified coverage tools, we analyzed the available reports to pinpoint areas with low coverage, particularly focusing on model-related files. We also look for code coverage badges (Codecov, Coveralls, Scrutinizer) and parsed the coverage reports we found.

RQ_{1d}(Test Suite Evolution and Maintenance): Studying the evolution of test suites in GitHub repositories provides valuable insights into the maintenance of test suites. Analyzing trends in the changes to test suites helps us pinpoint areas of code that have undergone substantial modifications, enabling a comprehensive assessment of software reliability.

This investigation includes observing the frequency of changes in the test suite, specifically concerning modifications to models or software components. Furthermore, the identification of changes, spanning additions, deletions, and modifications to test files, adds depth to

Table 4.2: List of the selected GitHub repositories

Framework	Project	Contributors	Stars	Forks
TensorFlow	https://github.com/GPflow/GPflow	77	1800	436
	https://github.com/conda/conda	424	5900	1400
	https://github.com/deepchem/deepchem	161	4600	1500
	https://github.com/XanaduAI/strawberryfields	48	725	186
	https://github.com/kubeflow/pipelines	28	3200	1400
PyTorch	https://github.com/snorkel-team/snorkel	71	5600	856
	https://github.com/open-mmlab/mmdetection	436	26000	9000
	https://github.com/Lightning-AI/lightning	842	25100	3100
	https://github.com/pyg-team/pytorch_geometric	448	18900	3400
	https://github.com/arraiyopensource/torchgeometry	240	8700	886
Keras	https://github.com/IBM/adversarial-robustness-toolbox	102	4000	1100
	https://github.com/dask/distributed	310	1500	704
	https://github.com/wandb/wandb	159	7500	561
	https://github.com/scikit-learn-contrib/imbalanced-learn	78	6600	1300
	https://github.com/stellargraph/stellargraph	32	2800	419

our investigation. Our analysis reveals how changes in a test suite are intertwined with the overall software development process.

To conduct this investigation, we selected 15 projects in total, selecting 5 for each framework (TensorFlow, Keras, and PyTorch), whose evolution has been analyzed in detail. The selection of the project has been driven by specific criteria to ensure the robustness and reliability of our analysis. To determine the projects, we selected the ones with the highest number of stars and forks, and those with model-related tests. Additionally, we prioritized repositories that provided a coverage report, ensuring a more comprehensive and insightful analysis for our study. By employing these criteria, we aimed to narrow down our focus to repositories that not only represent the popularity of a framework but also exhibit a strong testing infrastructure, contributing to the validity and depth of our research findings. Table 4.2 lists the selected projects with their respective parameters, including contributors, stars, and forks.

We start our examination from the commit history and pull requests, computing the percentage of commits related to tests. We leverage the GitHub Search API to identify commits and PRs related to testing. We search for commits and PRs containing the keyword 'testing' to retrieve relevant results. These indicators capture the degree of activity on the test cases, compared to the overall project activity.

To access commit messages, we used GitHub's API to extract the complete version control history for each selected repository. As several projects span hundreds of releases, analyzing every release is infeasible and highly time-consuming. To ensure a representative sample, we strategically selected 10 releases with uniform intervals based on each repository's full release history. This approach maintained consistent spacing between analyzed versions while still capturing meaningful evolutionary trends.

From the extracted data, we tracked the addition, modification, or deletion of test files and

calculated the size of the test suite for each selected project. Our purpose is to assess the overall growth of test suites within the selected repositories over time. With this analysis, we computed the total number of test files per version and the percentage of test files modified in each version.

When available, we finally tracked the code coverage metrics for each version to evaluate the impact of test suite growth on the thoroughness of code testing. Correlating test suite growth with code coverage trends would help to determine whether the increase in test cases is effectively improving or keeping steady code coverage.

4.3 Results and Analysis

In this section, we present findings for each research question, supported by quantitative data and qualitative observations.

4.3.1 RQ_{1b}(Test Case Presence and Types)

Out of 300 analyzed projects, 232 (77%) projects have a test suite, defined as collections of test files organized in dedicated directories or following naming conventions as shown in Table 4.3. This adoption ratio is substantially higher than previous studies of general open-source projects, though notably the gap narrows in more recent work: older empirical studies reported 37.5% adoption (Kochhar et al. [135]), while more recent investigations found 61.65% (Silva et al. [136]), suggesting that test adoption in open-source software has generally improved over time. Our finding of 77% adoption in DL projects indicates that DL developers have embraced testing practices more readily than traditional software developers, though it also reveals that nearly one-quarter (23%) of DL projects still lack any systematic testing, a concerning gap for safety-critical applications.

Even more concerning, only 128 of 232 projects with test suites (55%) have tests specifically aimed at exercising model components. This adoption of model-specific testing varies modestly across frameworks: 50% for TensorFlow (39 of 78 projects), 53% for Keras (40 of 75 projects), and 62% for PyTorch (49 of 79 projects). This trend aligns with Wang et al.'s [59] observation that 68% of sampled DL projects are not systematically tested at the model level, indicating that developers tend to prioritize testing infrastructure and data processing over validation of core model behavior and learned representations.

When considering the types of tests implemented within these repositories, our analysis revealed a predominant presence of functional tests, with a limited number of tests related to non-functional aspects, as reported in Table 4.4. Almost every project incorporates functional test cases, implemented as unit or integration tests, while system testing is less frequent.

Table 4.3: Test occurrence in the selected projects

Framework	Projects with Test Suite	No.of Untested Projects	Proportion of Projects with Model Tests
TensorFlow	78	22	39/78 (50%)
Keras	75	25	40/75 (53%)
PyTorch	79	21	49/79 (62%)
Total	232 (77%)	68 (23%)	128/232 (55%)

Table 4.4: Type of test cases implemented in Python GitHub projects

Types of Test	DL Frameworks			
	TensorFlow	**PyTorch**	**Keras**	**Total**
Unit	39	54	41	110 (37%)
Integration	17	12	10	39 (13%)
Smoke	-	5	3	8 (3%)
Performance	16	10	12	28 (9%)
System	1	-	3	4 (1%)
Security	1	2	-	2 (1%)

Non-functional tests seem less prevalent; our findings reveal that a small subset of projects, specifically 16 for TensorFlow, 10 for PyTorch, and 12 for Keras, incorporated performance tests. Additionally, we found that 5 PyTorch and 3 Keras projects contain smoke tests. Furthermore, we did not find any evidence of security testing in Keras, while only 1 TensorFlow and 2 PyTorch projects contain security tests. This imbalance suggests that DL developers prioritize functional correctness over non-functional properties. However, for safety-critical applications like ADS, performance, reliability, and security are equally important [14]. The lack of non-functional testing represents a significant gap in current practices.

There are several plausible reasons for this disparity, including developers' emphasis on core functionality, limited resources, and the inherent complexity of non-functional testing. Developers often prioritize functional testing because of its immediate value and relative ease of automation, as it focuses on verifying that the software performs its intended tasks correctly [209]. In contrast, non-functional testing typically demands greater time, resources, and specialized expertise. Moreover, non-functional properties such as performance and security are harder to quantify and automate, making them more challenging to test systematically than functional requirements [210].

To address this, projects should integrate non-functional requirements early, invest in automated tools for non-functional tests, and continuously monitor to ensure balanced attention to both functional and non-functional aspects of testing, leading to more robust software.

Table 4.5: Test automation frameworks

Automation Framework	DL Framework				
	TensorFlow	PyTorch	Keras	Total	Percentage
PyTest	34	54	41	129	48 %
Unittest	41	39	30	110	40 %
JUnit	2	4	4	10	3 %
Gtest	4	2	1	7	2.6 %
Jest	3	2	1	6	2.2 %
Tox	2	0	4	6	2.2 %
xUnit	1	0	0	1	0.37 %
TestNG	1	0	0	1	0.37 %
Mocha	0	0	0	0	0 %

4.3.2 $\mathbf{RQ_{1b}}$(Test Automation Adoption)

Our analysis revealed that a substantial portion of projects under investigation have adopted test automation. Automated testing adoption is identified in a project when it integrates at least one library related to test automation into its source code. As illustrated in the Table 4.5, 201 (86.63%) out of 232 projects adopt some form of test automation, to be precise, 67 in the case of TensorFlow, 65 for Keras, and 69 for PyTorch across various categories.

Results show that PyTest and UnitTest are the most used automation frameworks with percentages of 48% and 41%, respectively, consistent with their status as the most popular Python testing frameworks [187, 188]. PyTest is often preferred for its concise syntax, powerful fixtures, and rich plugin ecosystem, while UnitTest (Python's built-in framework) is often chosen for its simplicity and zero-dependency setup. The presence of non-Python frameworks (JUnit, GTest, Jest) targeting other languages indicates that many DL projects are multi-language and require combining multiple tools to achieve comprehensive testing. Overall, these findings suggest that DL developers rely primarily on mainstream Python testing frameworks rather than developing DL-specific frameworks.

Furthermore, our investigation also revealed that 53.6% (161 out of 300 projects) of the analyzed projects included GitHub workflow directories. However, only one-third included test automation (54 projects out of 161) within these workflows. This gap indicates that while automated testing is relatively common within the codebase itself, projects are far less consistent in integrating these tests into automated CI pipelines, suggesting a significant shortfall in end-to-end automation practices.

These findings indicate that although developers recognize the benefits of test automation, many do not integrate automated testing into their continuous integration (CI) workflows. Despite these challenges, automated test execution is critical for preventing regressions, particularly in collaborative projects where multiple contributors frequently introduce changes [74]. This gap between CI adoption and test automation highlights an opportunity

Table 4.6: Distribution of code coverage tools

Coverage Tool	Count	Percentage
Codecov	55	18.33%
Coveralls	13	4.33%
Scrutinizer	1	0.34%

for tool development, specifically for lightweight testing strategies, such as smoke tests or model inference tests that do not require retraining, that can be executed efficiently within CI constraints while still providing meaningful feedback on system behavior.

4.3.3 $\mathbf{RQ_{1c}}$(Code Coverage Analysis)

Our findings revealed that only 23% (69 out of 300) of the projects included coverage badges within their documentation. These badges provide links to the coverage tools, including Codecov [211], Coveralls [212], and Scrutinizer [213], offering access to detailed code coverage reports as illustrated in Table 4.6. Codecov emerged as the most popular choice, with 55 projects (18.33%) utilizing it. Coveralls followed with 13 projects (4.33%), and only one project (0.34%) used Scrutinizer. We also encountered 12 repositories containing files associated with coverage tools, but corresponding reports were missing, potentially indicating abandoned or incomplete setups.

Among the projects that reported coverage metrics, the overwhelming majority used **line/statement coverage** as their primary metric; no projects in our sample reported branch coverage, path coverage, or DL-specific coverage metrics (e.g., neuron coverage). Since the coverage metric type was effectively homogeneous (line coverage) across the reporting projects, we compared coverage percentages directly without normalization. However, a critical finding is that coverage measurement was typically applied to the entire codebase (including utility code, data pipelines, and infrastructure) rather than specifically to model-related code. Focusing on the projects with coverage reports, we observed that 19 (27.5%) did not provide any details specifically related to model test coverage. This translates to 72.5% of projects with reports including some level of coverage for model-related tests. However, a more concerning finding emerged when examining the broader pool of projects that included model-associated tests. Only 50 out of 128 (39%) such projects had associated coverage reports. This significant discrepancy suggests potential shortcomings in testing practices for the core deep learning models themselves. Further investigation is necessary to understand the reasons behind this gap and assess the overall effectiveness of testing across these projects.

To analyze the depth of code coverage, we categorized projects based on their reported percentages (presented in Table 4.7). The majority of projects (67%) achieved coverage above 80%, with 35% of the projects exceeding 90%. This distribution suggests that a

Table 4.7: Distribution of projects with code coverage percentage

Category	Count	Percentage
Below 50	4	5.80%
50-59	5	7.25%
60-69	3	4.34%
70-79	11	15.95%
80-89	22	31.88%
90-100	24	34.78%

significant portion of the projects target high coverage when projects invest in coverage measurement. Conversely, only a small number of projects fell into the lower coverage categories: 5.80% had coverage below 50%, and 27.54% were in the 50-79% range.

However, coverage percentages must be interpreted with caution. High code coverage does not necessarily imply effective testing, coverage measures which lines of code are executed, not whether the program's behavior is meaningfully validated [214]. In DL projects, this limitation is even more pronounced; coverage primarily reflects the execution of infrastructure code (Python functions, classes) rather than model behavior. A project might achieve 90% coverage simply by running data loaders and training loops without ever validating model predictions. This underscores the inadequacy of relying on code coverage alone as an indicator of testing quality in deep learning software. Developing DL-specific coverage metrics, such as neuron coverage [81], surprise adequacy [215], or decision boundary coverage [216] that better capture model testing effectiveness, remains an active research challenge.

4.3.4 $\mathbf{RQ_{1d}}$(Test Suite Evolution and Maintenance)

To investigate the evolution of test suites, we filtered projects (5 per framework) with more stars, model tests, and coverage reports available. We analyzed ten evenly distributed releases for each project from its version history, and monitored test file additions, modifications, deletions, and associated overall commit activity for the two controls.

Figures 4.1, 4.2, and 4.3 illustrate the results for TensorFlow, Keras, and PyTorch for test suite modification and test suite growth, respectively. Test suite modifications show the percentage of test files changed per release, while test suite growth shows the total number of test files per project from first to last release.

In particular, TensorFlow projects have higher test file modification rates per release, with changes in up to 100% of test files in some releases and a mean modification rate between 35–45%. Keras and PyTorch projects present less mutable rates: 15-40% for Keras, and 20–25% for PyTorch. The higher frequency of changes in TensorFlow projects may reflect its lower total count of test files compared to other frameworks. Further investigation

into the nature of the changes, project types, practices, and testing culture within each framework is necessary to obtain a clearer picture.

Despite varying modification rates, all the projects show an overall increase in the number of tests per version. Notably, some projects experience rapid spikes, growing a factor of 10x in 10 releases. The pace of release cycles can significantly impact this growth in test files. While the exact reason behind these spikes is unclear, possible explanations could be the restructuring of the codebase, the introduction of new features, bug fixes, or an increased emphasis on test quality to ensure reliability. This observation underscores the importance of regression testing: as test suites grow, effective test prioritization and selection strategies become essential for managing scalability. This need directly motivates the framework we develop in Chapter 6.

There is a natural relation between the total number of test files per project and the number of test files modified across versions in TensorFlow projects, as indicated in the plots 4.1. Projects with a higher number of total test files also tend to have a higher number of modified test files.

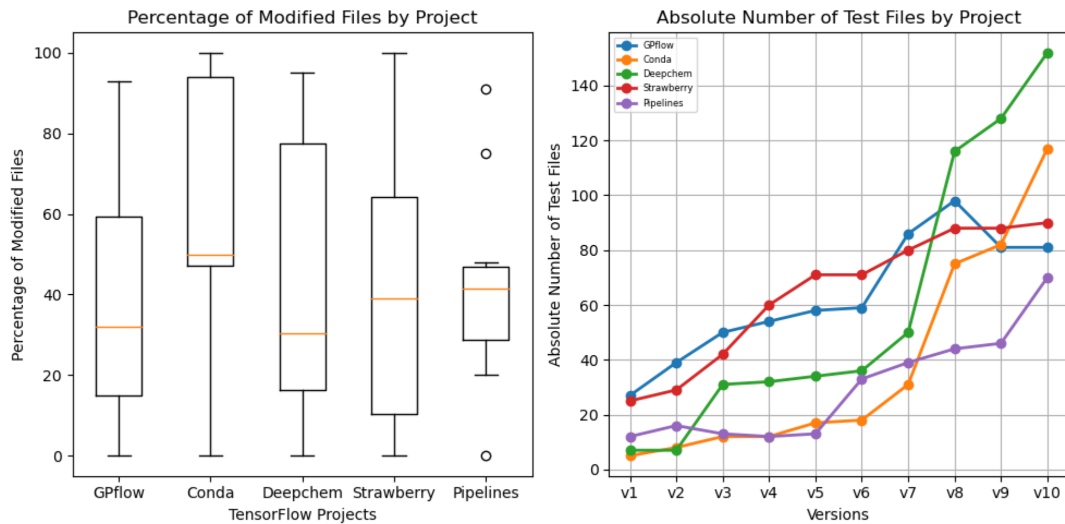


Figure 4.1: TensorFlow test suite growth.

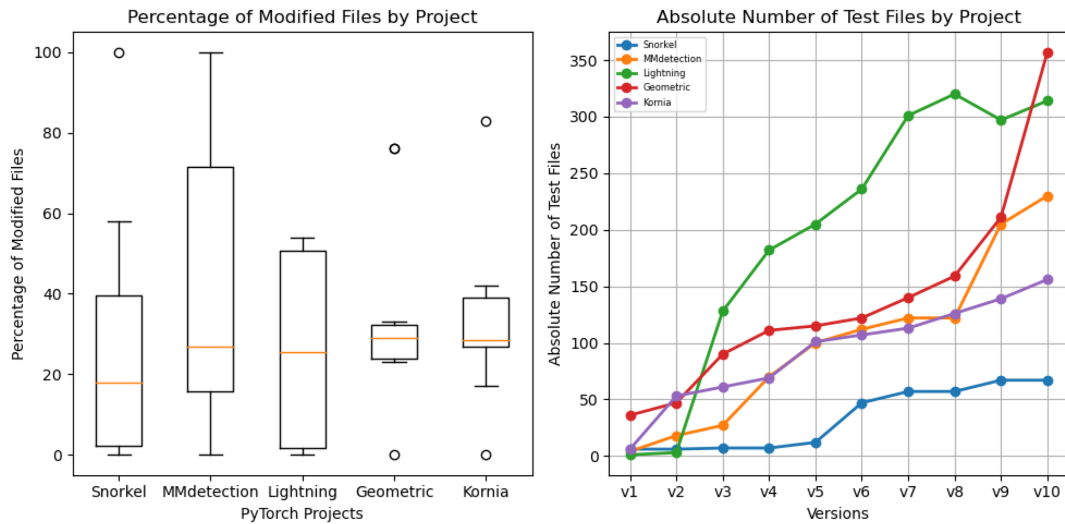


Figure 4.3: PyTorch test suite growth.

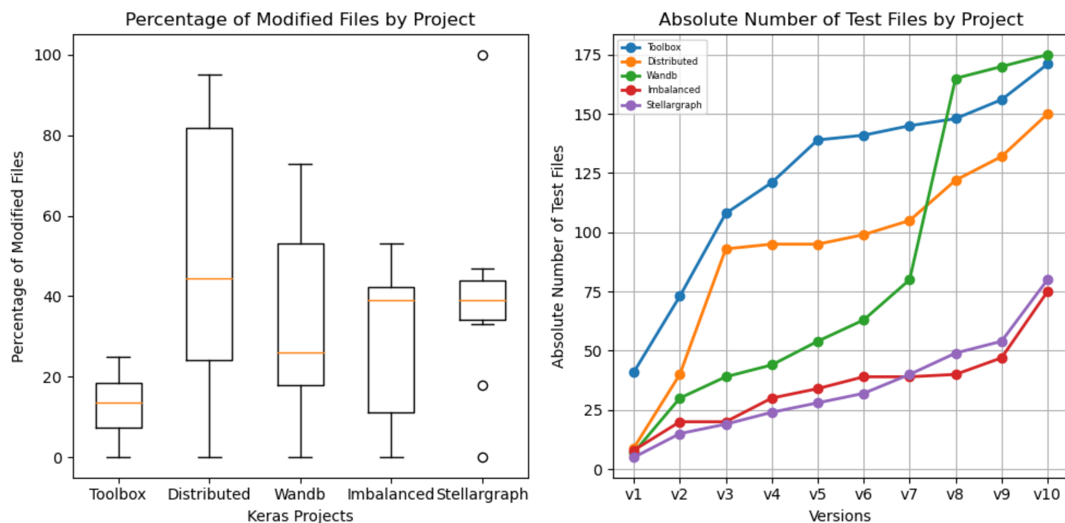


Figure 4.2: Keras test suite growth.

However, this relationship is not strictly linear. For example, the project with the largest test suite (Toolbox) does not exhibit the highest frequency of modifications between versions. Conversely, smaller projects, such as Stellargraph, with the least number of overall test files, show a substantial relative increase in the percentage of test files modified. This pattern suggests that the number of test files is not the only factor affecting the modifications; other factors, including project complexity, code churn, and the maturity of the testing infrastructure, may influence the frequency of changes.

Table 4.8 presents the results from our Commit and PRs analysis, aimed at understanding how developer communities discuss and enforce testing practices. Our findings reveal that TensorFlow projects have the highest average testing commit percentage (25%) while PyTorch has the lowest average, accounting for 11.7%. Interestingly, PyTorch has the

Table 4.8: DL framework commit analysis

Framework	Projects	Total No. commits	Testing commits	Per %	Total No. PRs	Testing PRs	Per %	Avg Testing Commit	Avg Testing PRs
TensorFlow	GPflow	2445	542	22%	1259	542	43%	25%	70.1%
	conda	16680	2000	12%	4036	1786	44%		
	deepchem	14916	3000	20%	14410	8000	56%		
	strawberryfields	1171	318	28%	607	314	52%		
	pipelines	5679	1000	18%	6827	5580	82%		
PyTorch	snorkel	2690	196	7%	742	395	53%	11.7%	72.7%
	mmdetection	2701	312	12%	3095	1515	49%		
	lightning	10084	1000	10%	9861	8659	88%		
	pytorch_geometric	7374	880	23%	2820	1523	54%		
	kornia	2642	598	23%	1747	1193	68%		
Keras	toolbox	12141	1000	8%	1310	898	69%	13.2%	66.3%
	distributed	5670	996	18%	4900	3135	64%		
	wandb	6073	858	14%	3954	3062	78%		
	imbalanced-learn	855	72	9%	491	191	39%		
	stellargraph	2535	378	15%	932	397	43%		

highest average of PRs related to testing, accounting for 72.7%, and Keras accounts for 66.3%, which is the lowest among the three frameworks. These differences likely reflect variations in community size, project maturity, and the availability of testing tools and frameworks tailored for each framework.

The high proportion of PRs mentioning testing indicates a strong culture of test-aware code review across the projects. Contributors are generally expected to include or update tests alongside code changes, and reviewers routinely evaluate the adequacy of these tests. This emphasis on testing during the review process explains the consistently high level of test adoption observed in RQ_{1b}.

4.4 Discussion

Our findings reveal a nuanced picture of testing practices in DL projects: adoption rates and automation are substantial, yet critical gaps persist in model-specific and non-functional testing. This section synthesizes findings, discusses their implications, and identifies areas for improvement.

4.4.1 Promising Trends

Our analysis reveals several encouraging trends that characterize a mature testing culture in deep learning software projects.

High test suite adoption: DL projects exhibit substantially stronger test suite adoption (77%) compared to general open-source Python projects reported in earlier empirical studies (37–62%) [135, 136]. However, it is important to note that testing adoption in open-source software has improved generally over time; the lower figures (37%) are from older studies [135], while more recent work reports higher adoption (62%) [136]. Thus, the elevated adoption rate in DL projects may reflect both a general improvement in testing culture across open-source communities and DL-specific factors such as industry

pressure for reliability in production systems, contribution guidelines in major frameworks (TensorFlow, PyTorch) that emphasize testing, and the maturation of Python's testing ecosystem.

Strong reliance on automated testing: The widespread use of test automation frameworks—primarily PyTest (48%) and Unittest (41%) demonstrates developers' commitment to reproducible and maintainable testing. Given the inherent complexity of DL pipelines (data preprocessing, model training, hyperparameter tuning, evaluation), automation is crucial for preventing regressions and ensuring continuous validation through CI/CD infrastructure.

Active and ongoing test maintenance: Testing is tightly integrated into development workflows. The high proportion of testing-related pull requests indicates that reviewers routinely scrutinize test sufficiency and expect contributors to provide or update tests alongside code modifications. This test-aware review culture significantly contributes to the overall robustness of DL development practices.

Continuous test suite expansion: The observed growth and modification of test suites across releases signal continued investment in quality assurance. Among the 15 projects analyzed for evolution, test suite growth ranged from 2x to 10x across 10 releases, with a median growth factor of approximately 4.5x. This sustained expansion, though introducing scalability challenges (longer execution times, potential redundancy), reflects a proactive approach to maintaining correctness as systems evolve and new features are added. The consistent test maintenance activity with 11.7–25% of commits related to testing across frameworks indicates that testing is integral to the development process rather than an afterthought. These observations motivate the test prioritization and reduction research presented in Chapter 6, which addresses the scalability challenges that emerge as test suites grow rapidly.

While these trends are encouraging, they must be interpreted cautiously. High code coverage does not necessarily equate to effective model testing, as established in RQ_{1c} findings: even projects with strong overall code coverage may have weak or absent coverage of model-specific behavior. Similarly, sustained test growth and maintenance reflect quantity rather than quality; rapid test accumulation without systematic prioritization or redundancy removal can create bottlenecks (as discussed in Chapter 6) rather than proportional improvements in fault detection. These trends demonstrate that DL projects have adopted testing processes aligned with traditional software engineering standards, yet significant gaps remain in the quality and focus of these practices.

These observations must also be understood in light of the study's methodological scope.

This chapter investigates testing practices along four dimensions, including test presence and types, automation adoption, coverage reporting, and test suite evolution, using large-scale static repository analysis. This design prioritizes scalability across 300 projects and focuses on consistently measurable artifacts, deliberately excluding dimensions that require dynamic analysis or external data sources. In particular, the study does not assess test quality, which would require computationally intensive techniques such as mutation analysis; it does not capture developer intent, which would necessitate qualitative methods such as interviews or issue analysis; and it does not examine the relationship between testing practices and software quality outcomes, which would require integrating repository data with defect and release histories under controlled conditions. Furthermore, while the dataset spans multiple deep learning domains, cross-domain variation in testing practices is not explicitly analyzed. These dimensions represent complementary directions for future work, building on the scalable baseline established in this study.

4.4.2 Critical Gaps

Insufficient Model-Specific Testing: Only 55% of projects with test suites include tests specifically targeting model components. This gap is critical in the sense that models are what make DL systems truly different from their traditional counterparts: the correctness, robustness, and fairness of these models directly dictate this system's reliability. Testing the infrastructure and data pipelines is essential, but not enough; the models should also be carefully evaluated with adversarial inputs, distribution shifts, and edge cases [183, 12].

Minimal Non-Functional Testing: Non-functional testing, including Performance (9%), security (<1%), and smoke (3%), is severely underrepresented. This is concerning for production deployments, where DL systems must handle large-scale data, maintain low latency, resist adversarial attacks, and gracefully degrade under failure. The scarcity of performance testing is particularly problematic for ADS, where real-time processing constraints are safety-critical [64].

Weak CI/CD Integration: Despite 53% of projects having GitHub workflows, only 18% overall integrate automated test execution into CI/CD pipelines. This suggests that many projects define workflows for build automation or deployment, but fail to enforce systematic testing on every commit. Without CI/CD-integrated testing, regression bugs may escape detection until late in development, increasing fixing costs and delaying releases [64, 205, 39].

Coverage Reporting Gap: 77% of projects lack any coverage reporting, leaving the thoroughness of their test suites unknown. Even more concerning, 61% of projects with

model tests do not report model-specific coverage. This blind spot may conceal untested code paths, particularly in complex model architectures where certain layers or loss functions may never be exercised by existing test suites [64, 217].

Test Suite Explosion: Test suites grow rapidly (up to 10× in 10 releases), yet few projects employ systematic test management strategies. Exhaustive execution of large test suites is time-consuming and computationally expensive, creating bottlenecks in continuous integration. This observation directly motivates our prioritization research (Chapter 6), where we develop behavior-aware test selection methods to identify the most valuable tests while minimizing execution cost.

The findings of this empirical study are directly consequential for the ADS testing methodology developed in the subsequent chapters. First, the substantial growth in test suite size mirrors precisely the scalability challenge encountered in ADAS regression testing, where large-scale benchmarks such as SensoDat already comprise over 32,000 scenarios, motivating test reduction and prioritization in (Chapter 6). Second, the persistent under-representation of model-specific testing confirms that the field has not yet converged on adequate strategies for testing learned components, which is the core problem motivating the ADS-specific methodology in Chapters 6 and 6. Third, the near-total absence of non-functional testing (security, robustness, performance) in open-source DL projects signals a maturity gap that is especially acute for safety-critical ADAS deployment. Taken together, these findings establish the need for more principled, behavior-aware, and scalable testing strategies.

4.5 Threats to Validity

This research is subject to limitations that could potentially influence the generalizability and conclusions drawn from the analysis of test suite growth and maintenance practices. Recognizing these threats to validity strengthens the transparency and trustworthiness of our findings.

Selection Bias: Focusing on popular repositories with high star and fork counts may bias results toward projects with stronger testing practices. Smaller or less popular projects may exhibit different practices. We mitigated this threat by selecting diverse projects across three frameworks (TensorFlow, Keras, PyTorch) and multiple domains (computer vision, natural language processing, reinforcement learning), as those most likely to influence community practices and be reused in production.

Data Extraction Limitations: Our reliance on keywords and naming conventions for identifying test-related activities may overlook cases where developers do not explicitly

mention testing. This could lead to an underestimation of the actual testing frequency. We mitigated this by supplementing manual inspection of project files to identify test-related activities within commits, even when not explicitly mentioned.

Generalizability: Findings from Python DL projects and selected frameworks (TensorFlow, Keras, PyTorch) may not generalize to other languages (Java, C++) or non-DL machine learning systems. However, Python is the dominant language for DL development (used in > 90% of DL projects), making our results representative of the broader DL ecosystem [218]. We attempted to mitigate this threat by clearly defining the scope and limitations of our study. Future studies could explore test suite growth patterns across a broader range of programming languages and frameworks to enhance generalizability.

Metrics Limitations: Counting test files or commit messages may not accurately reflect actual testing rigor. A project with 100 low-quality tests may be worse than one with 10 high-quality tests. We addressed this by triangulating multiple metrics (test presence, types, automation, coverage, evolution) to build a comprehensive picture.

Evolution Analysis Constraints: Analyzing only 15 projects (5 per framework) inherently limits the statistical power of our findings regarding evolution patterns; however, the consistency of findings across all studied frameworks (all showed rapid test suite growth) increases confidence in the conclusions. A more significant constraint lies in the validity threats introduced by measurement metrics. Simple quantitative metrics, such as counting test files, commit messages, or test-related pull requests, may not accurately reflect true testing rigor or quality (e.g., 100 low-quality tests vs. 10 high-quality tests). We attempted to mitigate this by using multiple, diverse metrics and cross-referencing data from different sources to build a comprehensive and validated picture of the test suite growth and activity.

The empirical study in this chapter provided a macroscopic view of testing practices in the deep learning ecosystem, revealing both encouraging trends and a clear need for more rigorous and scalable testing methodologies. However, general observations alone are insufficient for driving meaningful improvement. To make practical progress, we next move from ecosystem-level analysis to targeted intervention in a domain where inadequate testing has especially severe consequences.

The following chapter makes this transition by focusing on ADS, a safety-critical domain, where weaknesses in testing can translate directly into real-world hazards, amplifying the importance of methodological precision. Our first step addresses a foundational requirement for advancing ADS testing: *interoperability*. Without the ability to reuse, standardize, and transfer test scenarios across heterogeneous models and platforms, constructing diverse

and extensive test suites becomes prohibitively expensive, thereby contributing to the testing gaps observed in this chapter.

Chapter 5

OpenCat: An Interoperability Bridge for ADS Testing

5.1 The Scalability Challenge of ADS Benchmark

As demonstrated in Chapter 4, test suites in deep learning projects grow substantially (by up to 10×) as projects mature, thereby increasing the need for efficient regression testing. In ADS, this challenge is magnified by the availability of large-scale benchmarks such as SensoDat [40], which provides 32,580 road scenarios, DeepScenario [41] offers 80,000 driving scenarios, and SCTrans [42] contains 50,000+ scenarios derived from real-world logs. These benchmarks serve as essential tools for systematically assessing ADAS models, verifying their robustness under varied operational conditions, and conducting regression testing following model updates.

However, the effectiveness of these benchmarks for regression testing relies on a fundamental assumption that scenarios and their pass/fail labels are *reusable*, meaning they can be applied to evaluate different ADAS models across various simulation platforms without requiring their complete regeneration. When this assumption holds, researchers and practitioners can efficiently reuse existing benchmarks to validate new models.

In practice, they often focus on technical implementations that detail specific test executions for particular simulation platforms and ADAS models, instead of defining the test objectives in a general and reusable manner. As a result, each new model or simulator demands creating new benchmarks from scratch, which severely limits scalability and impedes cross-platform validation, because scenario definitions become tightly coupled to specific formats and technical implementations, making reuse across tools and models impractical [219, 149]. This chapter addresses this interoperability challenge by introducing OpenCat, a converter that bridges OpenDRIVE and Catmull-Rom spline representations, enabling large-scale benchmark reuse across simulators while empirically demonstrating the deeper problem of model-dependent scenario effectiveness, a finding that motivates behavior-aware testing methodologies in the subsequent chapter.

5.1.1 The Interoperability Challenge: Format Lock-in

Testing ADAS, such as lane-keeping functions (as described in section 2.3.3), requires creating road topologies or using predefined benchmarks. Recent years have seen a growing focus on testing methodologies for ADAS/ADS, particularly concerning test generation techniques and the proposal of benchmarks for ADAS testing [220, 221, 222, 223, 224, 225,

226, 227, 228, 229, 230, 231].

In this study, we investigate the reusability of ADS benchmarks by examining whether benchmark scenarios can be transferred across different road representation formats and simulation platforms while preserving geometric fidelity, and whether test case pass/fail labels derived from one ADAS model generalize to other models, or remain model-specific. The study focuses on SensoDat [40], a prominent benchmark comprising 32,580 road scenarios generated with test generators such as Frenetic [232] and AmbieGen [151]. SensoDat encodes scenarios in the OpenDRIVE format [123] and executes them in the BeamNG.tech simulator [30] using BeamNG's built-in AI driver (a PID-based controller with global knowledge of the road topology). Each scenario is labeled as a *pass* if the ego vehicle remains within lane boundaries (no out-of-bounds, OOB) and as a *fail* otherwise. Overall, SensoDat reports 19,926 passing tests (61%) and 12,654 failing tests (39%).

Assessing reusability requires overcoming a technical interoperability barrier: SensoDat's OpenDRIVE representation is incompatible with many research-oriented simulators. For instance, the Udacity Self-Driving Car Simulator [47], widely adopted in ADS testing research [115, 19, 60], represents roads using Catmull–Rom spline defined by waypoint sequences. These splines generate smooth, continuous, and computationally efficient trajectories for the vehicle control, but are fundamentally different from OpenDRIVE's representation. As a result, SensoDat scenarios cannot be executed directly in Udacity, motivating the need for a dedicated conversion tool.

To address this interoperability gap, we develop OpenCat, an open-source tool that converts OpenDRIVE road representations to Catmull-Rom spline representations. OpenDRIVE is widely used in industry for Level 3-4 ADS testing, while the Catmull-Rom spline is preferred in academia for Level 2 ADAS testing because of its computational efficiency and ease of manipulation. By applying OpenCat to the SensoDat dataset, we achieve 100% geometric accuracy ($R^2 = 1$), enabling the execution of SensoDat scenarios in academic simulators without loss of geometric fidelity. This conversion resolves the format lock-in imposed by OpenDRIVE and allows benchmark scenarios to be reused across simulation platforms. However, geometric interoperability alone does not guarantee behavioral or semantic equivalence across ADAS models, a challenge that we examine in the next section.

While some format converters exist (e.g., OpenSCENARIO to CommonRoad [153], OpenDRIVE to Lanelet2 [174]), none support the OpenDRIVE to Catmull–Rom transformation required for the Udacity simulator. Moreover, existing converters often trade geometric precision for other objectives such as routing efficiency or lane-level topology. However, in control-oriented testing, where steering decisions depend directly on high-fidelity road geometry, such compromises are unacceptable. Precise geometric preservation is therefore essential for reliable scenario reuse across simulators. While OpenCat resolves syntactic

interoperability, the following section shows that deeper semantic assumptions embedded in benchmarks remain unresolved.

5.1.2 The Semantic Challenge: The Model-Specific Brittleness

Even with perfect geometric conversion, a more fundamental issue remains: Do pass/fail labels transfer across different ADAS models? SensoDat's labels were produced using BeamNG's built-in AI driver, a PID-based controller with full, noise-free knowledge of the road topology (i.e., access to the exact reference line). In contrast, learning-based models such as Dave-2 [101] rely solely on camera input, must infer road geometry visually, and operate under partial observability.

We hypothesize that pass/fail labels are inherently model-specific: ADAS architectures differ fundamentally in how they perceive the environment and generate control actions (rule-based vs. learning-based, perception-based vs. end-to-end, feedforward vs. recurrent); consequently, they may fail on different road geometries due to variations in control strategies, error-recovery behaviors, and robustness. If this hypothesis holds, SensoDat's pass/fail labels are tightly coupled to BeamNG's AI driver and cannot be assumed to generalize to other ADAS models. Our findings confirm that pass/fail ratios diverge by 25-50% compared to the original SensoDat results, demonstrating that reliance on specific ADAS models impacts the interoperability of existing benchmarks, thereby calling for architecture-agnostic solutions to enhance their reusability and usefulness for regression testing and cross-model validation.

5.2 The OpenCat Conversion Approach

This section provides an overview of the road conversion process from OpenDRIVE to Catmull-Rom spline.

5.2.1 System Overview

Figure 5.1 depicts the conversion process from OpenDRIVE format to Catmull-Rom spline. The conversion process simplifies the complex road model to focus on its geometry, specifically its shape and path, while preserving elevation profiles and lane widths. The process comprises the following stages:

1. **Data Extraction:** OpenDRIVE road field is extracted from the SensoDat dataset scenario files (.xodr format). These (.xodr) files were then converted to OpenDRIVE JSON format used by OpenCat.
2. **Geometry Parsing:** We extract necessary geometry attributes, including position coordinates (x, y), starting position s, heading hdg, elevation profile, and length

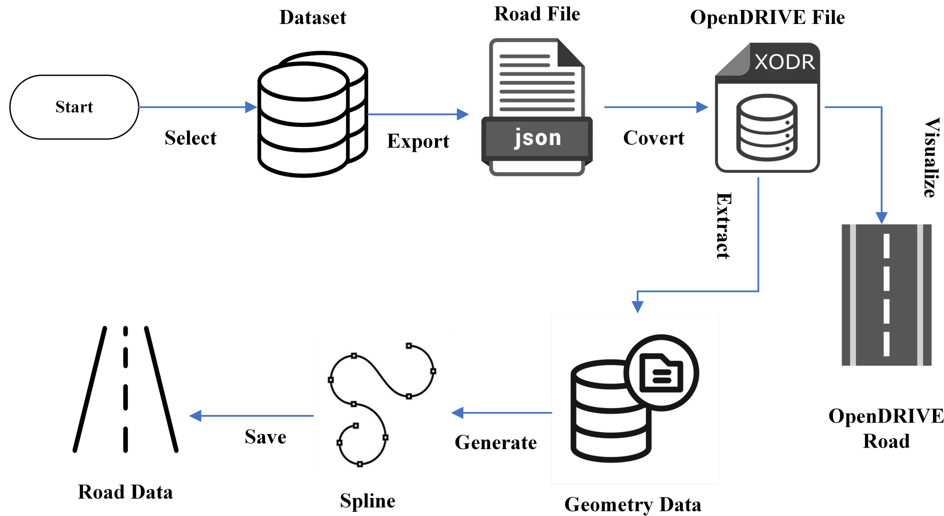


Figure 5.1: Conversion process from OpenDRIVE format to Catmull-rom spline.

from the OpenDRIVE XML structure.

3. **Reference Line Calculation:** The reference line, also known as the centerline (the path the vehicle should follow), is calculated from the OpenDRIVE data by averaging lane boundaries (right and left). In OpenDRIVE, the road network is built around a central reference line, which specifies how the road curves, bends, and transitions. All lanes and elevation profiles are linked to this reference line as shown in Figure 2.2a.

4. **Spline Generation:** Catmull-Rom splines require control points through which the curve interpolates. The control points are defined along the reference line to generate the Catmull-Rom spline, ensuring appropriate distribution to accurately capture the curvature of the road [233].

The entire process of data extraction and spline generation is detailed in the following subsections.

5.2.2 Conversion Algorithm

Algorithm 1 processes the OpenDRIVE file and extracts the road geometry for each lane side (right, left). For each road in an OpenDrive file, the algorithm iterates through its geometry sections to extract the attributes such as position (x, y), and starting position (s) by calling the auxiliary function *parse_geometry* (Line 5). Next, it calculates the elevation profile and width of the road at the current position *(s)* based on polynomial coefficients found in the OpenDRIVE data (Line 6). It then computes the lane offset (Line 7), which is the total width of the lanes from the road center, depending on the specified lane side (left or right). Based on the *lane_side* parameter, the algorithm adjusts

```
Algorithm 1: extract_road_geometry(opendrive, lane_side)
   Input: opendrive, lane_side
   Output: road_data
 1 road_data ← [];
   /* Initialize a list to store road data                                  */
 2 foreach road ∈ opendrive do
 3 |  planView ← road.planView;
 4 |  foreach geometry ∈ planView.geometries do
   |  |  /* Iterate through geometries of the road                          */
 5 |  |  (x, y, s) ← parse_geometry(geometry);
   |  |  /* Extract geometry attributes                                     */
 6 |  |  elevation_width ← get_elevation_and_width(road, s);
 7 |  |  lane_offset ← compute_lane_offset(road, s, lane_side);
   |  |  /* lane_side is either left or right                               */
 8 |  |  adjusted_x ← x + lane_offset.x;
   |  |  /* Adjust x-coordinate for lane                                     */
 9 |  |  adjusted_y ← y + lane_offset.y;
   |  |  /* Adjust y-coordinate for lane                                     */
10 |  |  append(road_data, (adjusted_x, adjusted_y, elevation_width.z, elevation_width.width));

11 return road_data;
```

the x and y coordinates by the computed lane offset (Lines 8-9), ensuring that only points from the desired lane side are extracted. Finally, the extracted road data *(x, y, z, width)* is collected (line 10) and returned (line 13).

Once the road geometry has been extracted, we proceed to generate the spline curve using *Algorithm 2*, which constructs a Catmull-Rom spline and interpolates points along the spline, ensuring that the road geometry is represented as a smooth curve. Details of the supporting functions used in this process are explained below.

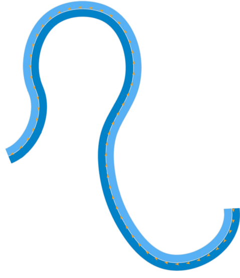
5.2.2.1 Lateral and Elevation Profile

The *parse_geometry* function extracts basic geometry attributes of a road from the OpenDRIVE XML structure, including *'x'* and *'y'* coordinates, *'length'* (the length of the geometry section), *'hdg'* (the heading or direction of the road at the starting point), and *'s'* (the position along the road where the geometry section begins). These values describe the shape and orientation of the road section, which is sufficient for generating a flat 2D plane that represents the road's horizontal geometry, but it does not capture the true 3D nature of the road terrain, such as slopes.

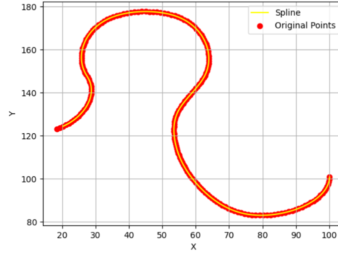
To address this, we use the *get_elevation_and_width* function to extract both the elevation and the width of the road at a specified position *'s'*, ensuring a more accurate road representation. If the current road has an associated elevation profile, it iterates over the elevation segments and calculates the elevation for the given position (s) using polynomial coefficients. For each segment, it checks whether the current position *(s)* lies within that segment by comparing *(s)* with the starting point of the elevation and the length of the

Algorithm 2: generate_spline(*opendrive, lane_side*)

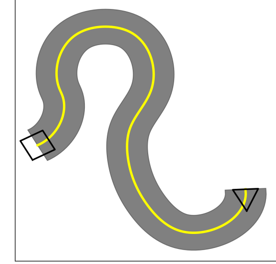
```
  Input: opendrive, lane_side
  Output: {control_points, spline_points} or empty_list
1 right_lane_points ← extract_road_geometry(opendrive, right);
  /* Extract right lane points                                  */
2 left_lane_points ← extract_road_geometry(opendrive, left);
  /* Extract left lane points                                   */
3 min_points ← 4;
  /* Minimum points for spline generation                       */
4 if len(right_lane_points) < min_points or len(left_lane_points) < min_points then
5     return empty_list;
6 control_points ← compute_centerline({right_lane_points, left_lane_points});
  /* Compute centerline                                         */
7 spline_generator ← compute_catmull_rom_spline({control_points});
  /* Generate spline                                            */
8 spline_points ← spline_generator.generate_spline();
9 return {control_points, spline_points};
```



(a) Original OpenDRIVE road.



(b) Generated spline with original points.



(c) Road in Udacity simulator.

Figure 5.2: Comparative view of OpenDRIVE road, generated spline, and test case.

segment. If *(s)* falls within this range, the function calculates the elevation at this position using a cubic polynomial formula:

$$z = a + b \cdot ds + c \cdot ds^2 + d \cdot ds^3 \tag{5.1}$$

Here *(a, b, c, d)* are polynomial coefficients, and *(ds)* is the difference between *(s)* and the start of the segment *(ds = s - s_elev)*.

After calculating the elevation, the function extracts the road's width by locating the lanes within the *laneSection*. For each lane in the section, it calculates the width using similar polynomial coefficients and sums the widths to get the total road width. The calculated elevation and width are then returned, providing the necessary data to represent the physical characteristics of the road at that position. This information is used to calculate the lane offset, which is explained below.

5.2.2.2 Lane Offset

Lane offset refers to the lateral (sideways) displacement of a lane or a vehicle path relative to a specific reference line. It is a critical concept in intersection safety, autonomous vehicle path planning, and standardized road representations like OpenDRIVE [234].

The *compute_lane_offset* function calculates this lateral position at a given point (s) along the road's reference line, accounting for lane width, curvature, and geometric variations. Accurate computation of lane offset is crucial for road modeling, simulation, and visualization, particularly when dealing with scenarios with varying road curvature or lane widths. First, we check the validity of each lane in the lane section on the road, skipping over the center lane *(lane_id = 0)*. For valid lanes, the width of each lane is calculated using polynomial coefficients, similar to the elevation and width calculations earlier. The width of each lane is then added to the *lane_offset*, which accumulates the total offset of all valid lanes for the specified side. Once all lanes have been processed, the total lane offset is returned.

Given this information, we now have enough details to generate a spline. To start with, the *Algorithm 2* retrieve road geometry data for the specified lane side (*right* or *left*) by invoking Algorithm 2 (Lines 1-2). These points represent the lane boundaries along the road. To ensure there are enough points to generate a spline, the algorithm verifies if both the right and left lane data meet the minimum point (i.e., 4) requirement (Lines 3). If the condition is satisfied, it computes the centerline by averaging the points of the right and left lanes, producing a set of control points that represent the middle of the road (Line 7). For well-formed roads where lanes are symmetric or near-symmetric about the reference line (a standard assumption in road design [235]), the arithmetic mean of corresponding right and left boundary points provides a geometrically sound approximation of the road centerline. Using these control points, the algorithm then initializes a *Catmull-Rom Spline* generator, configured with an *alpha=0.5* parameter to ensure balanced smoothness and to create one spline point per segment. Finally, the spline is generated, and both the control points and the generated spline points are returned (Lines 9-10).

5.3 Experimental Evaluation

To investigate whether existing ADS benchmarks can be made truly reusable across different simulators and ADAS models (RQ_2, Chapter 1), we applied our OpenCat converter to the publicly available OpenDRIVE scenario dataset SensoDat [40]. This empirical investigation addresses two complementary sub-questions:

$\mathbf{RQ_{2a}}$ **(Accuracy)**: *How accurately does* OpenCat *transform the OpenDRIVE road geometry into the Catmull-Rom spline representation while preserving geometric fidelity?*

RQ$_{2b}$ (Comparison): *How does the converted dataset perform relative to the original* SensoDat *dataset in terms of the test pass/fail ratio?*

5.3.1 RQ$_{2a}$ (Accuracy): Quantitative and Qualitative Validation

We converted a total of 32,580 roads from the SensoDat dataset using OpenCat. Before converting the OpenDRIVE files into the spline, we verified the validity and usability of the data in the OpenDRIVE files by visualizing the road structure using the OpenDRIVE file viewers [236, 237], as shown in Figure 5.2. Figure 5.2a presents an original road from the OpenDRIVE scenario visualized through the OpenDRIVE viewer [236], while Figure 5.2b presents the conversion result, where the generated spline (yellow) is overlaid with the original points from the OpenDRIVE file (red dots). The visual results confirm that the generated spline closely follows the original geometry, demonstrating the accuracy of the conversion. OpenCat preserves the original geometric characteristics of the road while producing a smooth representation suitable for use in simulations.

In addition to visual validation, we also verify our results using the following metrics to measure the precision of the generated spline by comparing the spline points with the original points.

Geometric Accuracy: The accuracy metric evaluates how closely the spline interpolation matches the original points by calculating the average Euclidean distance between the spline points and the original points. To enable fair comparison across roads of different geometric scales and extents, we normalize this average distance by the maximum possible error, defined as the diagonal distance of the bounding box enclosing all original points. This normalization yields a scale-independent accuracy metric ranging from 0% (worst case: spline deviates by the full diagonal distance) to 100% (best case: spline perfectly matches original points). The resulting percentage indicates how accurately the spline interpolation captures the original road geometry relative to the spatial extent of that road:

$$\text{Accuracy} = \left(1 - \frac{\text{Average Distance}}{\text{Max Possible Error}}\right) \times 100 \tag{5.2}$$

where *Max Possible Error* is the diagonal of the bounding box: $\sqrt{(\Delta x_{\max})^2 + (\Delta y_{\max})^2}$ for a road's x and y coordinate ranges.

R-square value: R-squared (coefficient of determination) quantifies the goodness of fit of the spline interpolation compared to the original data. It measures how well the spline points fit the original data. An R^2 value equal to 1 indicates that the spline fits the data well, while lower values signify a poor fit [238, 239].

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2} \tag{5.3}$$

where y_i are the actual values. $\hat{y}_i$ are the predicted/interpolated values and $\bar{y}$ is the mean of the original values. The conversion achieved perfect accuracy (accuracy = 100%) and R-squared (R^2=1), confirming the correctness and quality of OpenCat.

These results, combined with visual validation, demonstrate that OpenCat achieves *perfect geometric fidelity*: converted roads are indistinguishable from originals in terms of shape, curvature, and spatial layout.

5.3.2 RQ_{2b} (Comparison): Pass/Fail Ratio

We implemented and validated our spline-based roads for the Udacity simulator [47] using the test case generation framework by Biagiola et al. [120]. The framework employs a DAVE-2 (a convolutional neural network developed by NVIDIA) model as a system under test [52] [101]. DAVE-2 is an ADAS for lane-keeping that learns to drive by identifying latent patterns in a training dataset composed of images captured during expert human driving. We use a publicly available pre-trained model [52] to ensure reproducibility.

Crucially, DAVE-2 operates independently of the technologies underlying SensoDat, ensuring an unbiased evaluation of the converted benchmark. The original SensoDat benchmark was executed using BeamNG.tech's AI driver autopilot, a PID-based controller with global knowledge of road topology. To evaluate cross-platform and cross-model interoperability, we re-executed the converted scenarios using DAVE-2 in the Udacity simulator.

Test Case Generation: First, we initialize road scenarios by defining control points derived from our generated spline, similar to the test case representation in Biagiola et al. [120]. The control points represent critical aspects of road geometry from the OpenDRIVE file, such as lane boundaries, curves, and elevation changes. After interpolation, a road is considered valid if it meets certain criteria, which include: (1) the start and end points are distinct; (2) the road remains within a predefined square bounding box of 250 $\times$ 250 units; and (3) there are no intersections. Figure 5.2c presents an example of such a valid road generated by the Udacity test generator using our road data. We executed simulations on a MacBook Pro M2, 8 cores, 16 GB RAM, macOS Sequoia, and on an Intel Core i5, 12 cores, 16 threads, 4GB DDR6 RAM, NVIDIA GeForce RTX 3050 Ti, 16 GB, Windows 11.

Table 5.1 specifies the campaigns executed on either macOS or Windows, with (m) indicating macOS and (w) indicating Windows. This dual-system approach also helps further validate our methodology by testing the OpenCat's converted dataset performance

Table 5.1: Test pass/fail counts across campaigns.

Campaign	Pass	Fail	Total	Exe_Time h/m	OpenCat Pass%	SensoDat Pass%
Ambeigen Campaigns						
2_ambiegen(w)	963	10	973	5/19	98.97	58.17
3_ambiegen(w)	964	0	964	5/41	100	53.32
4_ambiegen(m)	956	9	965	5/50	99.06	51.30
5_ambiegen(m)	952	6	958	5/23	99.37	56.68
6_ambiegen(m)	951	8	959	5/44	99.16	54.12
7_ambiegen(m)	952	11	963	5/46	98.85	53.80
8_ambiegen(m)	941	11	952	5/46	98.84	54.31
9_ambiegen(m)	944	9	953	5/57	99.05	50.79
10_ambiegen(m)	964	7	971	5/38	99.27	53.88
11_ambiegen(m)	966	7	973	5/37	99.28	50.57
13_ambiegen(w)	944	10	954	5/34	98.95	53.85
14_ambiegen(m)	951	8	959	5/43	99.16	56.94
15_ambiegen(m)	939	13	952	5/33	98.63	54.20
Frenetic Campaigns						
2_frenetic(m)	920	8	928	5/38	99.13	60.56
3_frenetic(m)	944	10	954	5/36	98.95	63.34
4_frenetic(m)	956	8	964	5/44	99.17	65.06
5_frenetic(m)	935	10	945	5/39	98.94	66.67
6_frenetic(w)	854	90	944	5/33	90.46	65.93
7_frenetic(m)	958	9	967	5/53	99.06	64.33
8_frenetic(w)	859	93	952	5/37	94.01	63.87
9_frenetic(m)	959	5	964	5/51	99.48	64.20
11_frenetic(w)	828	28	866	5/17	96.76	63.51
12_frenetic(w)	928	28	956	5/38	97.07	63.07
13_frenetic(w)	925	34	959	5/38	96.45	67.06
14_frenetic(m)	857	9	866	5/10	98.96	68.59
15_frenetic(m)	860	10	870	5/5	98.87	61.84
Frenetic_v Campaigns						
2_frenetic_v(w)	943	1	944	5/51	99.90	67.63
4_frenetic_v(m)	524	1	525	3/16	99.80	65.52
5_frenetic_v(w)	940	0	940	5/49	100	67.45
6_frenetic_v(w)	763	1	764	4/42	99.86	66.58
7_frenetic_v(w)	47	0	47	0/18	100	55.32
11_frenetic_v(w)	949	4	953	5/58	99.58	65.27
12_frenetic_v(w)	888	54	942	5/53	94.26	66.24
13_frenetic_v(m)	946	5	951	6/1	99.47	70.97
14_frenetic_v(m)	928	6	934	5/44	99.35	72.93
15_frenetic_v(w)	925	24	949	5/58	97.47	64.18

on two different operating systems.

Test Oracle: We adopt the same oracle as SensoDat: a test case is considered successful if the vehicle completes the road without going out of bounds (OOB), defined as exiting the drivable lane area. In contrast, a test case fails if the car goes OOB (i.e., off the road) [115], or when the scenario itself is invalid (e.g., overlapping start and end points, as in road_818 from campaign_2_ambiegen), which prevents the simulation from initializing the vehicle. The Udacity simulator provides cross-track error (CTE), the lateral distance from the lane centerline to detect OOB. A vehicle is considered OOB if $|CTE| > 3.0$m (half the typical lane width of 6m), which is consistent with prior work [115].

Each scenario initializes the vehicle at the first control point, centered and aligned to the road direction to ensure consistent starting conditions. As the vehicle drives along the paths defined by the spline, we monitor lane-keeping via CTE and detect OOB events following established methods [120]. This enables systematic identification of safety violations during execution.

Table 5.1 presents the simulation results; we executed 32,580 tests in total. The results reveal a significant improvement in the ratio of passing tests across all test campaigns compared to the original SensoDat [40]. The table employs a color scheme to distinguish the passing ratio between OpenCat's and SensoDat. The green color indicates campaigns where OpenCat outperformed SensoDat with a percentage difference greater than 30%, the yellow color represents campaigns with a difference of 25% to 30%, and the red color indicates cases where the difference was less than 25%. Unlike the original dataset, where pass rates varied considerably, especially in challenging road structures such as Frenetic campaigns, our results show consistently high pass rates, with 32,035 passing tests and only 545 failing tests, resulting in an overall pass ratio of 98%, compared to a 61% in the original SensoDat dataset, which recorded 19,926 passing tests and 12,654 failing tests.

In particular, while the Frenetic campaigns in SensoDat exhibited high failure rates, our results demonstrate that an independent ADAS model achieves substantial success rates. Even the most challenging campaigns are achieving over 95% pass rates, with only a few notable exceptions (i.e., *campaign_6_frenetic* and *campaign_8_frenetic*. Ambiegen campaigns, which performed poorly in the original SensoDat dataset, with pass rates ranging between 50% and 58%, performed exceptionally well in our evaluation, consistently exceeding a 98% pass rate. Moreover, macOS and Windows executions yielded nearly identical results, highlighting the robustness of our evaluation. Similarly, Frenetic_v campaigns, which already showed strong performance in SensoDat, achieved near-perfect pass rates in our simulations.

In addition to safety outcomes, we incorporated execution time as an evaluation metric

(Table 5.1, column 5), providing insight into the computational efficiency of different campaign types, which is missing in the original SensoDat paper. Execution time was measured as the wall-clock duration from scenario initialization (vehicle positioned at start) to completion (vehicle reaching road endpoint or exceeding time limit). Cumulative execution time per campaign was aggregated across all scenarios to assess the relative computational burden. This information is valuable for practitioners planning large-scale simulation-based testing campaigns, enabling informed resource allocation decisions and highlighting which road geometries or test generation strategies impose the greatest computational cost.

Overall, our findings demonstrate substantial divergence in test pass rates compared to SensoDat and expose a fundamental limitation of existing ADAS benchmarks. Although the converted geometry is identical to the original (as established in RQ_{2a}, with $R^2 = 1$), the 25–50% difference in test pass/fail ratios cannot be attributed to conversion errors but rather reflects the interaction of multiple confounding factors: (1) the ADAS model architecture (end-to-end learning in DAVE-2 vs. PID-based control in BeamNG AI), (2) differences in simulator physics engines and rendering pipelines (vehicle dynamics, sensor noise models, lighting effects), and (3) the test oracle definition itself: BeamNG AI operates with global road knowledge and different lane-keeping tolerances than DAVE-2, which relies solely on visual input [31] (more details in the Section 5.4). These confounders collectively demonstrate that test criticality is not an intrinsic property of the scenario but depends fundamentally on the ADAS and simulator through which it is evaluated.

These results establish that benchmarks such as SensoDat are implicitly coupled to the specific ADAS model and simulator used during their creation and validation, severely limiting their applicability for evaluating alternative architectures or cross-platform regression testing. The independently evaluated ADAS (DAVE-2) successfully navigates 98% of scenarios in Udacity, compared to only a 61% pass rate in the original SensoDat context, demonstrating that model-specific pass/fail labels are not portable across architectures. This finding underscores the inadequacy of using historical pass/fail annotations for regression testing when ADAS systems evolve or when scenarios are transferred to different models, a critical insight motivating the behavior-aware prioritization framework in Chapter 6, which identifies scenario properties that generalize across diverse ADAS architectures rather than relying on model-dependent failure history.

5.4 Discussion and Implications

Our findings have profound implications for benchmark design and regression testing practices in ADS. This section interprets results, explores root causes, and proposes actionable recommendations.

5.4.1 Pass/Fail Differences

The 61% to 98% pass rate shift raises a critical question: *Why do Udacity's Dave-2 and BeamNG's AI driver produce such different outcomes on geometrically identical roads?* We identify the following contributing factors:

Simulator Differences: BeamNG.tech uses soft-body physics with deformable vehicle models and high-fidelity tire dynamics [30], while Udacity employs rigid-body physics with simplified dynamics [47]. This difference substantially affects vehicle handling: BeamNG vehicles exhibit more realistic slip, understeer, and oversteer behavior, making them genuinely harder to control on demanding roads. Udacity vehicles respond more smoothly and predictably to steering inputs, effectively providing easier control dynamics [240]. Crucially, this ease of control in Udacity likely inflates DAVE-2's pass rates relative to what would be observed in more realistic (or more difficult) physics simulations; consequently, the 98% pass rate should not be interpreted as absolute confidence in DAVE-2's robustness, but rather as a demonstration of model-specific performance under Udacity's simplified physics.

Additionally, rendering fidelity also diverges: BeamNG delivers photorealistic visuals (i.e., ray tracing and physically-based materials), while Udacity uses comparatively basic Unity rendering. Although rendering quality primarily affects perception models (image classifiers), it may also affect control indirectly: BeamNG's richer visual detail may introduce distractions such as shadows and reflections that degrade perception, whereas Udacity's simpler graphics reduce perceptual ambiguity [241, 242]. This could partially explain DAVE-2's high success rate; the cleaner visual environment may align well with the training data distribution (which likely contains relatively clear driving scenes), whereas BeamNG's photorealism introduces out-of-distribution visual artifacts that trigger perception errors.

Architectural Differences (Learning vs. Rule-Based Control): BeamNG's AI driver is a *rule-based* PID controller that combines navigation graphs (NavGraphs) for global route planning with PID loops for local vehicle control. It can access the full road reference line ahead, enabling perfect anticipation of upcoming curves [30]. Despite this informational advantage, it exhibits conservative behavior (only 61% passing ratio) and often fails even on moderately challenging roads. Dave-2, in contrast, is a *learning-based* end-to-end model that relies only on camera images [243]. It must infer road geometry from visual input under partial observability, with no explicit road /lane geometry or global path planning knowledge beyond the visible horizon [101]. Despite this limitation, Dave-2 succeeds on 98% of roads, indicating that imitation learning yields robust steering behaviors.

This apparent paradox, where a perceptually limited learning model outperforms a globally

informed rule-based controller, suggests that Dave-2's learned representations generalize better to diverse geometries than BeamNG's hand-tuned PID parameters. An alternative explanation is that BeamNG's AI optimizes for passenger comfort (i.e., minimizing lateral acceleration) over OOB avoidance, which can cause failures on roads that require more aggressive steering [244, 245].

Oracle Threshold Sensitivity: Our out-of-bounds (OOB) threshold ($|\text{CTE}| > 3.0\text{m}$) may differ from SensoDat's threshold. Although both rely on lateral displacement from the lane centerline as the failure metric, SensoDat's documentation does not explicitly specify the precise threshold triggering failure. Even small threshold differences (e.g., 2.5m vs. 3.0m) could introduce discrepancies, though they are unlikely to fully account for the observed 37 percentage-point difference. More substantially, the definition of *lane boundaries* may differ between simulators: BeamNG may enforce stricter lane boundaries than Udacity, or may penalize even temporary excursions, whereas Udacity may tolerate brief boundary contact. These oracle subtleties compound the architectural and physics differences identified above.

These observations collectively demonstrate that scenario criticality, whether a road causes ADAS failure, is not an intrinsic property of the road geometry but emerges from the interaction of: (1) ADAS architecture and control philosophy, (2) simulator physics and rendering, and (3) oracle definitions. Therefore, benchmarks labeled with outcomes from one model cannot reliably predict performance on alternative models or simulators. This finding directly addresses $\text{RQ}_{2\text{b}}$ and establishes that technical interoperability (geometric conversion) is necessary but insufficient for semantic interoperability (consistent test meaning) across ADAS architectures.

Consequently, our study highlights the need for architecture-agnostic ADAS benchmarks that define test success in terms of observable, model-independent safety properties rather than model-specific outcomes. Designing such benchmarks and identifying scenario properties that generalize across diverse control architectures and simulators remains an open research challenge and a critical direction for advancing ADAS validation practices, as explored in subsequent chapters.

5.4.2 OpenCat as an Enabler for Cross-Platform Testing

While OpenCat enables full technical interoperability by converting OpenDRIVE road descriptions into Catmull–Rom splines with 100% geometric fidelity, our empirical results show that this alone is insufficient to ensure benchmark portability. Specifically, semantic equivalence—the preservation of a scenario's safety-critical characteristics and failure-revealing potential across different ADAS models and simulators is not guaranteed by format conversion. Despite identical road geometries, pass/fail outcomes vary substantially

(from 61% to 98%), highlighting the influence of differences in ADAS architectures, simulator physics, and oracle calibration. This divergence reflects model-specific brittleness that limits the portability of instance-based pass/fail labels, and suggests that scenario criticality emerges from the interaction of these factors rather than being solely an intrinsic property of the road itself [149].

Beyond achieving lossless OpenDRIVE–to–Catmull-Rom conversion, OpenCat provides critical infrastructure that enables deeper empirical analysis of ADS testing. First, it facilitates *multi-simulator* and *cross-model* evaluation, allowing the same benchmark scenarios to be executed in environments such as BeamNG and Udacity, as well as across learning-based ADAS models such as DAVE-2 and Chauffeur. This capability exposes model and simulator-specific brittleness, helping distinguish genuine model weaknesses from platform artifacts.

Second, OpenCat enables *Coverage-based test selection* by making it possible to collect rich behavioral traces (e.g., speed, steering, CTE) across large-scale scenario benchmarks. These data, combined with geometric features, form the basis of the coverage-guided clustering and prioritization framework developed in Chapter 6, which moves beyond instance-level pass/fail labels toward architecture-agnostic regression testing.

More broadly, the applicability of OpenCat extends beyond the specific tool chain evaluated in this thesis. It enables researchers to reuse OpenDRIVE-based benchmarks across lightweight simulators with minimal engineering effort, supports test generation tools in improving interoperability across scenario formats, and provides a foundation for designing future benchmarks with multi-format compatibility. While the current implementation focuses on a single conversion direction and target simulator, its modular architecture, comprising parsing, geometry extraction, and serialization components, supports extension to additional formats, representing a key direction for increasing its adoption and impact.

Consequently, OpenCat's contribution extends well beyond benchmark portability: it is an *infrastructure layer* for scalable, cross-platform ADS testing, enabling systematic investigation of model-specific brittleness and supporting the development of principled, behavior-aware testing methodologies.

5.5 Threats to Validity

This research on benchmark interoperability and cross-platform scenario reuse is subject to limitations that could potentially influence the generalizability and conclusions drawn from our analysis of OpenCat conversion accuracy and ADAS model-specific performance variation. In this section, we address potential limitations and threats to the validity of our findings.

Conversion Accuracy and Geometric Fidelity: A potential concern is that discrepancies in pass/fail outcomes may arise from errors introduced during the OpenDRIVE-to-Catmull-Rom transformation. Although spline fitting and discretization could theoretically introduce deviations, our quantitative validation (RQ_{2a}) demonstrates that conversion is highly accurate: geometric accuracy is 100%, with ($R^2 = 1$) across all converted scenarios. While minor numerical precision errors (e.g., floating-point rounding in coordinate transformations) may introduce imperceptible deviations at the sub-millimeter scale, these are negligible compared to typical steering corrections required for lane-keeping [231]. Visual inspection further confirms perceptual equivalence between original and converted roads. Critically, if conversion artifacts were responsible for the observed pass/fail divergence, we would expect degraded performance. Instead, we observe a substantial increase in pass rates (from 61% to 98%), directly contradicting this hypothesis and indicating that conversion fidelity is not the primary cause of model-dependent performance variation.

Oracle Definition and Threshold Dependency: The out-of-bounds (OOB) criterion used to classify pass/fail may differ between SensoDat and our evaluation. Our choice ($|CTE| > 3.0$m) aligns with half a standard lane width and prior work [120], but SensoDat does not explicitly document its threshold. Even substantial threshold variations (e.g., ± 0.5m) would not plausibly account for the observed 37 percentage-point difference (61% vs. 98%). Such modest oracle differences might explain small discrepancies ($\pm$5–10%), but not the magnitude observed, suggesting that oracle definition is a minor contributor compared to architectural and simulator differences.

Simulator Physics and Pass-Rate Inflation: The Udacity simulator employs simplified rigid-body physics compared to BeamNG's soft-body dynamics, potentially inflating DAVE-2's success rate (98%) by making road control artificially easier. We acknowledge this limitation explicitly: the observed pass rate should be interpreted as DAVE-2's performance under Udacity's specific physics, not as an absolute measure of robustness that would necessarily transfer to higher-fidelity simulators. However, this simulator-specific bias does not invalidate the core finding: identical road geometries yield markedly different pass/fail outcomes across simulators and ADAS models, demonstrating that benchmark labels are environment- and model-dependent. Even if Udacity's simplified physics contributes to higher pass rates, the fundamental conclusion remains unchanged.

Semantic Preservation and Test Oracle: OpenCat abstracts away OpenDRIVE semantic elements not essential for lane-keeping testing, such as lane markings, traffic signs, surface materials, and elevation profiles. For perception-heavy ADAS tasks (e.g., sign recognition, object detection), this abstraction would be inappropriate. However, for control-focused lane-keeping tasks, where steering performance is primarily driven by

road geometry (curvature, lane width), the abstraction is justified. Additionally, we adopt out-of-bounds detection as the primary test oracle, which effectively captures catastrophic failures but may miss subtler safety concerns such as steering smoothness degradation or excessive acceleration. However, OOB events are strong indicators of unsafe lane-keeping behavior [246, 247], and this choice aligns with prior work [149, 120].

Limited Generalization Across Simulators and ADAS Models: Our empirical evaluation focuses on SensoDat scenarios executed with a single learning-based ADAS model (DAVE-2) in the Udacity simulator. While this limits direct generalization to other simulators (CARLA, BeamNG, LGSVL) or alternative ADAS architectures, the scale and diversity of the benchmark (32,580 scenarios across three generation campaigns) provide strong evidence within the lane-keeping domain. Importantly, Chapter 6 extends this investigation to a second architecturally distinct ADAS model (Chauffeur, employing LSTM layers), demonstrating that model-dependent performance variation persists across learning-based architectures. This multi-model validation provides evidence that the observed brittleness is not unique to DAVE-2.

This chapter investigated the portability of ADS benchmarks by introducing OpenCat, a tool for converting OpenDRIVE road networks into Catmull-Rom spline representations, and by empirically evaluating the reusability of SensoDat scenarios across different simulators and ADAS models. Our results show that Model-specific brittleness fundamentally limits the generalization of instance-based pass/fail labels, even when geometric fidelity is perfectly preserved. These findings motivate a paradigm shift in benchmark design, from instance-based labels toward pattern-based specifications, where roads are characterized by geometric and behavioral properties. This shift directly motivates the prioritization framework developed in Chapter 6, which clusters scenarios based on geometric and behavioral similarity, selects representative cases to ensure pattern coverage, and prioritizes them based on geometric complexity, dynamic difficulty, and historical failure information, thereby enabling more robust cross-model generalization.

Chapter 6

Coverage-Guided Road Selection and Prioritization for Accelerated Testing of Autonomous Driving Systems

Successful conversion of the large-scale SensoDat dataset in Chapter 5 exposes a new challenge that threatens to undermine the practical value of interoperability: the scalability problem. Gaining access to SensoDat's converted 32,580 scenarios and continuously published benchmarks from the research community, a conservative estimate suggests that modern ADAS evaluation pipelines must contend with tens of thousands of test cases. Empirical execution data from Chapter 5 quantify the magnitude of this burden: executing the complete SensoDat benchmark required around 100+ hours of continuous simulation per ADAS version. Evaluating multiple ADAS models or performing regression testing across multiple benchmarks increases the per-cycle testing cost, making exhaustive replay infeasible in fast-paced development cycles that require frequent validation.

6.1 Scalability Challenge

As simulation-based validation becomes central to ADAS development, the volume of available road scenario datasets has expanded rapidly. Contemporary benchmarks easily include tens of thousands of scenarios, many generated automatically through systematic test generation techniques. Replaying all scenarios exhaustively is increasingly impractical, particularly as systems evolve through model retraining or architectural changes, necessitating frequent regression testing. For development teams with limited computational resources or tight release schedules, 100+ hours per test cycle is prohibitive, forcing difficult choices between incomplete testing and delayed deployment [72, 74].

Crucially, not every scenario contributes equally to fault detection. Many scenarios are *redundant*, exhibiting similar geometric properties or inducing similar driving behaviors despite superficial differences. For example, two roads with different absolute lengths but nearly identical curvature profiles will induce nearly identical steering control actions, providing redundant validation value. Similarly, behavioral redundancy occurs when roads of different shapes nonetheless produce similar vehicle trajectories, speed profiles, and control outputs (e.g., a sharp curve and a long, gentle curve both requiring sustained lateral acceleration). Indiscriminate test replay executes these redundant scenarios unnecessarily, wasting computational resources and delaying discovery of truly critical failure-inducing scenarios. This inefficiency undermines the practical value of regression testing: if critical scenarios are not prioritized, they may be discovered late in the testing process or skipped

entirely when execution time exceeds available budgets, potentially leaving dangerous faults undetected until deployment [79, 73, 248].

Classical regression test prioritization methods focus on source-code coverage and change history, but are ill-suited to the ADAS domain, where behavior emerges from real-time interaction between the learning-based driving model and complex, continuous environments. As a result, the value of a test case cannot be adequately captured by traditional software metrics. Instead, scenario value is best understood in behavioral terms: test cases that induce challenging, unstable, or novel control responses are likely to expose failures more than nominal driving scenarios, while redundancy, both geometric and behavioral, should be systematically identified and eliminated to streamline test suites without sacrificing fault-detection capability [249, 79, 81, 82, 83].

To address this scalability challenge, this chapter introduces a behavior-aware, coverage-guided prioritization framework that jointly considers road geometry and dynamic driving features. Coverage-guided selection ensures that the reduced test suite represents the full diversity of geometric and behavioral patterns present in the original dataset, maximizing the variety of road shapes, control challenges, and driving behaviors tested while minimizing redundancy. This approach contrasts with arbitrary test reduction, which may inadvertently remove critical scenarios, or with random selection, which offers no guarantee of representative coverage. Under practical resource constraints, exhaustive validation is infeasible; therefore, we must systematically determine: (1) which test scenarios to execute to maintain comprehensive coverage, (2) in what order they should be run to maximize early fault detection, (3) whether behavioral features (speed variability, steering variability, cross-track error) provide additional discriminative value beyond geometric analysis alone, and (4) whether prioritization strategies generalize across different ADAS architectures or remain model-specific. Addressing these questions enables practitioners to reduce test execution time while preserving fault-detection capability and ensuring that selected tests are representative of the full operational domain.

The proposed framework addresses these questions by clustering scenarios based on geometric and behavioral similarity, selecting representative patterns, and prioritizing cases with higher fault-revealing potential. In doing so, it transforms large, unwieldy scenario collections into compact, high-value test suites, preserving coverage while substantially reducing execution cost and enabling efficient regression testing across diverse ADAS models. In a nutshell, instead of treating all roads equally, our framework directs testing toward those segments most likely to expose failures.

Our framework operates on existing datasets, constructing a behavioral profile for each road derived from trajectory-level metrics such as steering variability, speed variability, cross-track error, and yaw rate variability. Based on these behavioral profiles, we cluster

redundant roads, eliminate those offering no novel insight, and rank the remaining segments according to their likelihood of revealing failures. A test case is considered a failure if the car goes OOB (i.e., off the road) [115].

We validate our framework using state-of-the-art imitation learning ADAS models across multiple driving environments, capturing diverse road geometries and driving dynamics. We compare our behavior-aware prioritization against three baseline strategies: (1) *Random replay*, where scenarios are executed in arbitrary order; (2) *Geometric-only prioritization*, which ranks tests based solely on road curvature complexity and curve severity; and (3) *Dynamic-only prioritization*, which ranks tests based solely on behavioral metrics (steering variability, speed fluctuations, cross-track error) without considering geometric properties.

Our framework substantially outperforms these baselines by ranking failure-inducing test cases significantly higher in the execution order, enabling substantially earlier fault exposure. In several environments, the first critical failure is identified up to 95× earlier in terms of executed tests, and overall testing effort is reduced by more than 80% without loss of failure coverage. More importantly, beyond accelerating first-failure detection, our approach concentrates the majority of failure-inducing scenarios within the top-ranked portion of the test suite, allowing practitioners to expose critical faults early in the regression testing process.

Our study highlights a broader implication for ADAS regression testing: evaluation should be driven by behavioral value rather than only dataset completeness. By focusing on behavioral difficulty, developers can accelerate risk discovery, reduce computational effort, and produce stronger evidence of system safety.

6.2 Behavior-aware Prioritization

Test prioritization is traditionally concerned with ordering test cases so that those most likely to reveal faults are executed earlier [250]. In classical software engineering, prioritization strategies often rely on coverage metrics, change history, or past failures to rank tests. The goal is not to eliminate tests, but to improve the efficiency of fault discovery under resource constraints. Early execution of high-value tests reduces time-to-detection and accelerates developer feedback, which is critical in iterative deployment cycles.

Transferring this principle to ADAS introduces unique challenges [32]. Road tests are not isolated function calls; they represent continuous trajectories over time, influenced by control dynamics rather than discrete logic. As a result, conventional source-based prioritization techniques, such as statement coverage or mutation impact, are not applicable as they fail to capture what makes a road test valuable [251]. Instead, value must be derived from behavioral evidence: a road segment that induces high steering oscillation or

lateral instability carries more fault potential than a visually complex but behaviorally trivial one [77].

We conjecture that effective prioritization for ADAS requires a shift from visual or input-based diversity toward behavior-based selection. Roads should not be distinguished by how different they appear as a whole, but by how different segments influence the driving model's control outputs. This reframes prioritization around driving challenge, using fine-grained metrics [77] that capture difficulty, instability, or the potential for misbehavior. For example, rather than treating a gentle 100-meter left curve and a sharp 50-meter left curve as equally valuable simply because they have different visual appearances or lengths [252], behavior-aware prioritization recognizes that both produce similar steering demands and vehicle responses (sustained lateral acceleration, moderate steering angle corrections). Conversely, a sharp curve followed by an immediate straightening maneuver may induce different control dynamics (abrupt steering reversals, yaw instability) than a gradual curve [252], warranting prioritization despite similar geometric complexity. This fine-grained, behavior-focused approach enables more intelligent test selection than uniformly replaying the entire dataset or arbitrarily sampling test cases.

The underlying intuition for our approach is rooted in a fundamental observation about test value: when two road segments induce nearly identical driving behaviors, similar steering angles, speed profiles, cross-track error patterns, or acceleration profiles, they provide redundant validation of the ADAS model. Executing both segments offers negligible additional insight into the model's robustness or potential failure modes beyond executing one of them. Conversely, road segments that challenge the model differently, requiring sharper steering, maintaining tighter lane adherence, or demanding rapid control corrections, expose different aspects of the model's learned behavior and are more likely to reveal distinct failure modes. Therefore, effective test prioritization must identify and prioritize behavioral diversity rather than assuming that visually or structurally distinct roads automatically induce substantively different driving behaviors.

Our approach recasts test minimization and prioritization as a behavior-aware coverage selection problem: first, reduce the test suite by clustering road segments exhibiting similar geometric and dynamic driving behaviors, then rank and prioritize the remaining tests based on geometric difficulty (curvature complexity, curve severity), behavioral instability (steering variability, speed fluctuations), and historical failure information (which segments previously induced failures) [79, 83]. This strategy ensures both that the selected test suite represents the diversity of driving challenges present in the original dataset and that high-priority tests are those most likely to expose critical failures early in the testing process.

Building on this reliance on historical behavioral data, the framework proposed in this

chapter is designed primarily for regression testing scenarios, where ADAS models are iteratively updated and re-evaluated against an existing scenario suite. In such settings, behavioral traces from prior executions can be directly leveraged by the prioritization mechanism, enabling substantial efficiency gains after an initial full execution of the benchmark. While this one-time cost is non-trivial, it is amortized across subsequent regression campaigns, where the framework achieves significant reductions in testing effort. For newly developed models without prior execution history, behavioral features are not yet available, and the framework operates in a geometric-only mode. As shown in Section 6.4.5, this baseline remains effective, outperforming random selection, though it is less efficient than the full hybrid approach. This distinction between initial evaluation and regression contexts is central to understanding the framework's applicability and expected performance in practice.

The proposed framework segments roads at fine granularity, reflecting the observation that redundancy often arises at the sub-road level rather than across entire scenarios. Similarity is therefore assessed using a combined representation of geometric properties (e.g., shape and curvature) and observed behavioral responses of the ADAS (e.g., speed, steering, cross-track error, and yaw), allowing structurally similar road segments that elicit different control behaviors to be distinguished. Central to this approach is the distinction that geometric diversity does not necessarily imply behavioral diversity: road scenarios that differ in length, curvature, or layout may induce nearly identical control behavior and thus yield redundant testing outcomes, while geometrically similar segments may present markedly different challenges depending on perception quality, control stability, and recovery mechanisms of the system under test. Consequently, scenario difficulty is not an intrinsic property of road geometry alone but emerges from the interaction between a scenario and a specific ADAS architecture, making behavioral difficulty inherently model-relative. Geometry-centric coverage metrics, therefore, risk overestimating test adequacy by emphasizing structural variation without accounting for system response. By incorporating execution behavior into similarity assessment and test selection, the framework prioritizes scenarios that are most likely to expose new or recurring failures, enabling more informative coverage and improving fault discovery efficiency in regression testing settings where historical behavioral data is available.

6.3 Framework Architecture

Our framework is organized as a pipeline to extract, compare, cluster, and prioritize road sections, also called *segments* (i.e., straight, curved) from existing whole tests, aiming to reduce fine-grained redundant test scenarios while preserving the geometric and behavioral diversity of road networks.

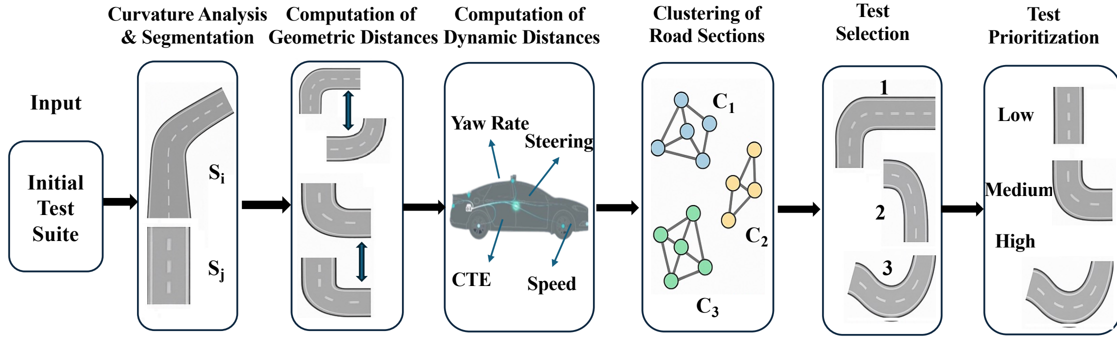


Figure 6.1: Coverage-based road selection and prioritization framework.

As illustrated in Figure 6.1, it starts with curvature-based road segmentation, decomposing roads into different section types (e.g., straight and curved) based on their geometric features. After that, it performs geometric matching using DTW [50] to identify similar patterns across roads while enforcing curvature consistency (left vs right side). DTW measures similarity between two temporal or sequential data of potentially different lengths, which reflects how well one sequence can be warped to align with another [50]. In addition to geometric features, our framework incorporates dynamic driving data (e.g., speed, steering angle, yaw rate, and cross-track error) extracted from simulation logs to enable more informative and realistic clustering of road sections. This data is collected by executing each road scenario in the Udacity simulator using the ADAS model under test (DAVE-2 and Chauffeur), recording the vehicle's behavior throughout the traversal. The resulting behavioral feature vectors complement geometric information, allowing similarity assessment to reflect actual model behavior rather than geometry alone.

These combined features jointly drive agglomerative clustering, which groups similar curves with similar ADAS behavior into compact clusters while preserving unique geometries. Once clusters have been formed, the next step involves coverage-based road selection, which identifies a minimal subset of roads that collectively represent all clusters. The selection rule is: for each cluster, select one or more representative roads (typically those exhibiting the cluster's characteristic geometric and behavioral properties) such that the union of selected roads covers all identified clusters. This ensures rare or unique patterns are included while minimizing fine-grained redundancies; roads within the same cluster are sufficiently similar that executing one representative provides coverage of that cluster's behavior space.

Finally, our framework applies a prioritization mechanism that ranks the selected roads and organizes the testing execution strategy. Selected roads are assigned to a high-priority queue and ranked based on geometric diversity (curvature complexity, curve severity), dynamic behavior difficulty (steering variability, speed fluctuations), and historical failure data (which roads previously induced failures). The most valuable and challenging scenarios execute first, enabling early detection of critical failures. Unselected roads are retained

Table 6.1: Framework configuration parameters

Parameter	**Symbol**	**Default**	**Rationale**	**Sensitivity Range**
Curvature threshold	τ_c	0.015 m^{-1}	Highway design standard for significant curves [235]	0.010–0.025 m^{-1}
Hysteresis window	w	3 points	Balances noise reduction vs. responsiveness to geometric transitions [253]	2–5 points
Min. segment length	L_{min}	10 meters	Minimum meaningful control sequence for ADAS evaluation [254]	5–20 meters
Length ratio threshold	τ_{len}	0.8	Discriminates between direct comparison vs. inclusion matching [255]	0.7–0.9
Dynamic weight	w_{dyn}	0.5	Equal contribution from geometric and dynamic features	0.0–1.0
Prioritization weights	α, β	0.5, 0.5	Equal emphasis on geometric complexity and dynamic difficulty	$\alpha + \beta = 1$
Historical failure bonus	$H(r)$	0.25	Fixed bonus for previously failed roads	0.1–0.5

in a deferred queue (low-priority queue) for potential future testing during extended or overnight execution windows, allowing practitioners to maintain comprehensive coverage when computational resources permit, while ensuring that high-value tests execute under normal time constraints.

Our framework relies on several configurable parameters that influence segmentation granularity (curvature threshold), clustering behavior (feature weighting), and prioritization scoring (criterion weights). While we provide recommended default values based on established literature and empirical validation [235, 253, 254, 255], practitioners may need to adjust these parameters based on their specific testing objectives, available computational resources, and risk tolerance. A sensitivity analysis examining the impact of parameter variations on test suite reduction and fault retention will enable practitioners to understand performance trade-offs when deviating from defaults. Table 6.1 presents the key configuration parameters used throughout our framework.

6.3.1 Segmentation and Curvature Analysis of Roads

Road segmentation transforms continuous geometric paths into sequences of discrete, geometrically homogeneous sections that enable meaningful comparison across scenarios, while curvature provides the fundamental geometric property for segmentation because it captures essential road shape characteristics that determine control difficulty for autonomous systems.

6.3.1.1 Initial Test Suite

The initial test suite $T = \{T_i\}$ consists of tests composed of two main parts $T_i = (R_i, C_i)$, where R_i is the road and C_i is the configuration of the scenario that has to be simulated on the road R_i. A *road* R_i is represented as an ordered sequence of Catmull-Rom spline points [149]:

$$R = \{p_1, p_2, \ldots, p_n\}, \quad p_i = (x_i, y_i) \tag{6.1}$$

where (x_i, y_i) are the coordinates along the road's centerline. The configuration C_i includes

the information necessary to run the simulation, such as the initial position of the vehicle and its speed.

6.3.1.2 Curvature Calculation

For each road R, we calculate the curvature κ value at each point (x_i, y_i) along the road, generating a continuous sequence of curvature values κ_i that characterizes the road's geometry. Curvature plays a central role in our study, as it captures the essential shape of the road, indicating where it is straight (near-zero curvature values), where it bends (right or left depending on the sign of the curvature value), and how sharply it turns (depending on the absolute curvature value). The curvature calculation employs a *three-point discrete approximation* based on the circumcircle radius of consecutive trajectory points [256]. For three consecutive points $p_{i-1} = (x_{i-1}, y_{i-1})$, $p_i = (x_i, y_i)$, $p_{i+1} = (x_{i+1}, y_{i+1})$, we first compute the determinant of the consecutive segment vectors to determine the turning direction:

$$\det = (x_i - x_{i-1})(y_{i+1} - y_i) - (y_i - y_{i-1})(x_{i+1} - x_i) \tag{6.2}$$

where $\det > 0$ corresponds to a counterclockwise (left) turn, while $\det < 0$ corresponds to a clockwise (right) turn, and $\det = 0$ indicates collinear points (straight line). Then the curvature magnitude *Rad* is derived from the circumcircle radius of the three points, which is calculated using the formula [257]:

$$Rad = \frac{|p_i p_{i+1}| \cdot |p_{i-1} p_{i+1}| \cdot |p_{i-1} p_i|}{2|det|} \tag{6.3}$$

where $|p_i p_j|$ denotes Euclidean distance between points i and j. The actual curvature is $\kappa = 1/Rad$, where $\kappa = 0$ represents a straight line (when $\det = 0$, resulting in $Rad \to \infty$). This approach provides computationally efficient curvature estimation, capturing both the sharpness of the curve (through curvature magnitude κ) and direction (through the sign) of road curves, which is essential for tasks such as trajectory planning and vehicle control.

To divide roads into smaller, geometrically homogeneous sections for fine-grained focused testing (straight, left curve, right curve), we use segmentation. Segmentation enables fair comparison across roads; without it, comparing two entire roads directly would be misleading, since roads can vary greatly in length and overall shape. Moreover, segmentation highlights distinctive road features and creates a meaningful unit (road section, also called a segment) for similarity matching, clustering, and prioritization.

6.3.1.3 Segmentation

Having computed curvature values $\{\kappa_1, \kappa_2, \ldots, \kappa_n\}$ at each control point, the segmentation process must identify transition points where road characteristics change qualitatively from straight to curved or between curve directions. The curvature-based segmentation process first employs a *hysteresis-based thresholding* framework [253] to categorize the shape of the road at each point by also considering the follow-up points, and then creates segments based on the computed information.

Given a sequence of w curvature values $\{\kappa_i, \kappa_{i+1}, \ldots, \kappa_{i+w-1}\}$ starting at point p_i (i.e., κ_i is the curvature value of point p_i), the shape $s_i = shape(p_i)$ of the road at position p_i is determined as follows.

- *straight*, if $\forall j = 1 \ldots i + w - 1, |\kappa_j| < \tau_c$,
- *left-curve*, if $\forall j = 1 \ldots i + w - 1, \kappa_j > \tau_c$,
- *right-curve*, if $\forall j = 1 \ldots i + w - 1, \kappa_j < -\tau_c$,
- all other cases, retain the classification of the previous point.

where w is the length of the window, and τ_c is the threshold value for straight sections. We use a curvature threshold of $\tau_c = 0.015m^{-1}$ (corresponding to an $R = 66.67m$) since it has been reported to effectively distinguish between geometrically significant curves that require substantial steering input and nearly-straight sections that can be traversed with minimal control adjustments [235]. We use a hysteresis window $w = 3$ to reduce noise while maintaining responsiveness to genuine geometric transitions, following established principles for discrete curve analysis [253]. Smaller windows respond quickly to curvature changes but risk noise sensitivity; larger windows provide stronger noise suppression but may blur transition boundaries or fail to detect brief curve segments [258].

From a sequence of shape values s_i, with $s_i \in \{straight, left\text{-}curve, right\text{-}curve\}$, segmentation has to establish the boundaries of each section. This is done by extracting subsequences of maximal length $\{s_j, \ldots s_{j+k}\}$ with homogeneous shape, i.e., $s_j = s_{j+i} \forall i = 1 \ldots k$ (e.g., a left-curve section where all its points are classified as left-curve). A minimum section length constraint of 10 meters is applied to avoid the generation of trivial sections [254]. Segments falling below this threshold are treated as a noisy section and merged into the preceding section, ensuring that each final section represents a meaningful geometric unit. This merging operation occasionally creates segments with slightly heterogeneous shape classifications near boundaries, but maintains the principle that segments should correspond to substantial road features rather than transient characteristics [259].

The final output is a *sequence of non-overlapping sections* $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ where each segment S_i includes its control points (start and end indices), segment type (straight

or curved), total length, and the curvature profile $\{\kappa_k : p_k \in S_i\}$. Curved sections are of particular importance, as they capture critical geometric features such as corners and turns that substantially influence driving behavior and vehicle dynamics, while straight segments provide context but are less discriminative.

Through the segmentation process, we reduce complex road geometries into discrete, analyzable units with consistent geometric properties. These segments serve as the foundation for subsequent geometric comparison, behavioral analysis, and clustering operations. Decomposing roads this way enables comparison at the appropriate granularity: matching curved sections against one another, identifying short curves within longer stretches, and acknowledging that a road with multiple distinct features should not be treated as an indivisible entity.

6.3.2 Computation of Geometric Distances

As a result of segmentation, we obtain many road sections, but not all of them are unique. Some sections may repeat across different roads (e.g., straight sections, identical curves) without contributing to exercising additional behaviors of the ADAS. We perform geometric matching to preserve distinctive geometries by identifying redundant sections and filtering out repeated patterns. In particular, we compare sections pairwise using DTW [50] on their curvature. Straight sections are flagged for coverage but excluded from similarity matching since they lack distinctive shape patterns. However, during the coverage-based road selection phase, straight sections are explicitly included as coverage targets: the selection algorithm ensures that at least one road containing each straight section type is represented in the final selected test suite. By focusing on the curved segments (right/left curves), we speed up the matching process and reduce the computational load while retaining the most discriminative road features, and maintaining full coverage of all road section types (including straights) in the prioritized test set.

DTW is a standard algorithm for measuring similarity between two sequences of potentially different lengths by performing non-linear temporal alignment, producing a distance (the difference between two curvature values) that reflects geometric similarity [50]. In practice, DTW ranges from 0 (identical sections) to 1 (completely different sections). Since we look for redundancies in the set of roads to be exercised, we are not only interested in nearly-identical sections but also in the inclusion between segments. In fact, if a section is included in another one, the shorter section would represent a redundant scenario compared to the longer one. To capture this case, we distinguish how we compute the similarity between sections of similar lengths from the matching for inclusion between sections.

In particular, given two sections, P and Q, their length ratio is defined as $lr(P,Q) = \frac{min(|P|,|Q|)}{max(|P|,|Q|)}$. If $lr(P,Q) < \tau_{len}$, $sim(P,Q) = 1 - DTW(P,Q)$, that is, if the two sections

have similar lengths, their similarity can be computed by using DTW directly. DTW handles segments of similar length effectively, but for segments with substantially different lengths, we must consider inclusion relationships explicitly. We empirically determined $\tau_{len} = 0.8$ as a good threshold to discriminate between sections that can be compared directly and sections that must be compared for inclusion [255].

For segments with substantially different lengths where (i.e., $lr(P, Q) \geq \tau_{len}$), the similarity between sections P and Q is computed by checking if the shortest section, namely P, is included in the longest one, namely Q. Note that if P is included in Q, using Q for testing implies having already tested the road geometry in P, but the opposite is not true.

The inclusion of P in Q is checked by considering every possible alignment between the two sections. In particular, for each alignment position k between 1 and $|Q| - |P|$, the similarity score of the considered alignment is defined as:

$$sim_k(P, Q[k : k + |P|]) = 1 - DTW(P, Q[k : k + |P|]) \tag{6.4}$$

where:

- $Q[k : k + |P|]$ denotes the subsequence of Q of length $|P|$ starting at index k.
- $|P|$ is the number of points in P.

Using a sliding-window framework, the shorter section P is aligned at every possible position k along the longer section Q, and DTW is computed for each alignment, ensuring we identify inclusion even when the matching subsequence appears in the middle or end of the longer segment rather than at its beginning. The final similarity between P and Q is given by the best similarity value computed at various alignment points. This value represents the degree of inclusion of one road into the other, in terms of its geometry. More formally, this is defined as:

$$sim(P, Q) = max_{k=1...|Q|-|P|} sim_k(P, Q[k : k + |P|]) \tag{6.5}$$

Finally, the distance function can be derived from the similarity as follows: $d_{geom} = 1 - sim(P, Q)$.

For each section pair, the process may yield a full match, one or more non-overlapping subsequence matches, or no match. All confirmed matches are stored bidirectionally (both sections reference each other) in the registry, including metadata on alignment ranges, similarity scores, and parent segment IDs. Segments without valid matches remain flagged as unmatched and indicate unique geometry. This matching procedure substantially improves the discovery of recurring geometric patterns (e.g., identical curves appearing in different roads) while preserving any unique road shapes.

6.3.3 Computation of Dynamic Distances

Geometric similarity captures important aspects of scenario relationships but provides incomplete information for prioritization [260]. To better capture scenario difficulty beyond road geometry, we augment geometric analysis with behavioral features extracted from simulation logs that characterize how autonomous systems respond to each segment during test execution. These behavioral features encode functional demands that segments impose independent of their geometric appearance, providing complementary information about scenario difficulty and testing value [19, 261]. We used this information later in clustering, creating groups of sections that reflect similarity both in geometry and in driving behavior.

An important practical consideration is that collecting dynamic driving data requires an initial execution of the full test suite, which may seem to offset the benefits of test reduction. However, this step is necessary for baseline validation and for generating the behavioral profiles used in test prioritization. The benefits emerge in subsequent regression cycles: as ADAS models are retrained or updated, only a prioritized subset needs to be executed. Given that the full SensoDat benchmark requires 100+ hours per run, and our approach achieves up to 89% reduction, each cycle saves roughly 90 hours of simulation time. Over multiple iterations, these cumulative savings significantly outweigh the one-time cost of initial data collection. For cold-start scenarios without prior execution data, we also provide a geometry-only prioritization mode that enables meaningful, though reduced, test reduction based solely on road geometry.

For each road segment S, we associate a set of dynamic features $D(S) = \{di_i^S\}$ that capture driving behavior and control difficulty. Specifically, speed variability $di_1^S = \sigma(\{v_t\}_{t\in S})$ measures the standard deviation of vehicle speed over time, reflecting planning complexity, perception uncertainty, or control instability.

The steering variability $di_2^S = \sigma(\{\theta_t\}_{t\in S})$, captures lateral control demands, as segments requiring frequent steering corrections exhibit higher variability. Prior work shows that elevated steering variability correlates with lane-keeping difficulty and increased failure likelihood [262, 28].

The mean cross-track error $di_3^S = \frac{1}{|S|}\sum_{t\in S}|cte_t|$, quantifies navigation precision by averaging lateral deviation from the lane centerline. While some deviation is unavoidable due to control delays and environmental disturbances, sustained errors indicate difficulty in maintaining the desired trajectory [263].

Finally, yaw rate variability $di_4^S = \sigma(\{\dot{\psi}_t\}_{t\in S})$, measures the variability of the vehicle's angular velocity about its vertical axis. Smooth yaw evolution reflects stable directional control, whereas high variability indicates oscillatory behavior or corrective maneuvers associated with reduced stability [264, 265].

Here:

- v_t = speed at time t,
- θ_t = steering angle at time t,
- cte_t = cross-track error at time t,
- $\dot{\psi}_t$ = yaw rate at time t,
- $\sigma(\cdot)$ = standard deviation operator.

Together, these features characterize segment difficulty across complementary dimensions of autonomous driving: planning and speed regulation, lateral control effort, trajectory tracking accuracy, and directional stability. Since the features operate on different physical scales, we apply min–max normalization independently to each feature:

$$\mathrm{di}_i^S(\mathrm{norm}) = \frac{\mathrm{di}_i^S - \min_S \mathrm{di}_i^S}{\max_S \mathrm{di}_i^S - \min_S \mathrm{di}_i^S}$$

where the minimum and maximum are computed across all segments. This normalization ensures comparability while preserving relative ordering and avoiding distributional assumptions inherent in z-score normalization [266].

We compute the distance between the dynamic indicators of two sections, P, Q, as follows.

$$d_{\mathrm{dyn}}(P, Q) = \frac{1}{4} \sum_{i=1}^{4} \left| di_i^P - di_i^Q \right| \tag{6.6}$$

This yields a distance value in [0, 1], with 0 indicating identical dynamic behavior and 1 maximal dissimilarity.

6.3.4 Clustering of Road Sections

Having computed both geometric similarity and behavioral distance, the framework proceeds to identify groups of similar segments that represent recurring patterns in the test suite. Clustering serves two purposes: it identifies redundancy by grouping segments that share both similar geometries and similar behavioral profiles, and it establishes coverage requirements by identifying distinct geometric-behavioral patterns that the selected test subset must represent.

We employ agglomerative hierarchical clustering [51] with *complete linkage*, a bottom-up approach that begins with each segment as its own cluster and iteratively merges the most similar clusters until an optimal structure emerges [51, 267]. Complete linkage is defined as the maximum distance between any pair of elements across two clusters: $d_{\mathrm{complete}}(C_i, C_j) =$

$\max_{x \in C_i, y \in C_j} d(x, y)$, where $d(x, y)$ is the pairwise hybrid distance combining geometric similarity (via DTW on curvature sequences) and dynamic similarity (via Euclidean distance on normalized behavioral feature vectors). This linkage criterion produces compact, well-separated clusters by requiring that all pairs of elements across merging clusters remain relatively similar [268]. Alternative linkage criteria like single linkage (minimum distance) risk creating elongated *chain clusters*, while average linkage provides intermediate behavior but lacks the interpretability of complete linkage's maximum-distance guarantee [269].

Agglomerative clustering offers several advantages over alternative approaches for our application [51, 267]. Unlike k-means clustering, which struggles with elongated or irregular cluster shapes and requires the number of clusters to be specified a priori [270, 271], agglomerative methods determine cluster count automatically based on data structure. In contrast to density-based approaches like DBSCAN [272], which identify clusters through local density and may label sparse regions as noise, agglomerative clustering assigns every segment to a cluster. This property is particularly important in road testing scenarios, where rare or outlier segments may correspond to critical corner cases that should be retained rather than discarded. Such segments typically appear as singleton clusters that merge only at high dissimilarity levels. Finally, unlike partitioning methods that produce a single flat clustering, agglomerative clustering yields a dendrogram encoding the complete merge sequence, enabling analysis at multiple levels of granularity if required [271].

Agglomerative clustering with complete linkage exhibits a time complexity of $O(S^2 \log S)$ for S segments, requiring $O(S^2)$ space for the pairwise distance matrix. Although computationally more expensive than linear partitioning methods like k-means, the inherent benefits, specifically automatic cluster count determination and the interpretability of the dendrogram structure, justify the expense for our application.

Sections are compared using a distance function that combines distances computed using geometric and dynamic data into a unified hybrid measure:

$$d_{hybrid}(S_i, S_j) = (1 - w_{dyn}) \cdot d_{geom}(S_i, S_j) + w_{dyn} \cdot d_{dyn}(S_i, S_j) \tag{6.7}$$

where $d_{\text{geom}}(S_i, S_j) = 1 - \text{sim}_{\text{geom}}(S_i, S_j)$ converts geometric similarity to distance, and $w_{\text{dyn}} \in [0, 1]$ controls the relative weight of behavioral versus geometric information. The weight parameter $w_{\text{dyn}} = 0.5$ provides balanced integration of both information sources, treating geometric and behavioral similarity as equally important.

Crucially, clustering operates within segment types rather than across the complete segment collection. We perform separate clustering for each segment type (i.e., straight segments, left-curve segments, and right-curve segments), recognizing that segments of different types address fundamentally different aspects of driving patterns and should not be considered

similar regardless of their specific geometric or behavioral characteristics. For example, a straight section, regardless of its length or how it's traversed, does not substitute for testing curve navigation. This type-based organization ensures that the selected test subset includes representatives from each category of road features [273].

The algorithm constructs a pairwise distance matrix D where entry $D[i, j]$ contains $d_{\text{hybrid}}(S_i, S_j)$ for segments S_i and S_j of the same type. The algorithm terminates when no pair of clusters has an inter-cluster distance below an automatically determined threshold $\tau_{cluster}$. Rather than imposing a fixed distance threshold or target cluster count, we employ the inconsistency method that analyzes the dendrogram to identify significant jumps in merge distances indicating natural cluster boundaries [274]. The method adapts the clustering threshold based on data variability using the Coefficient of Variation (CV):

$$CV = \frac{\sigma(D)}{\mu(D)} \tag{6.8}$$

$$\pi(CV) = \pi_{\max} - \frac{\text{clip}(CV, CV_{\min}, CV_{\max}) - CV_{\min}}{CV_{\max} - CV_{\min}} \times \pi_{\text{range}} \tag{6.9}$$

$$\tau_{\text{cluster}} = Q_{\pi(CV)}(D) \tag{6.10}$$

where:

- $\sigma(D)$ and $\mu(D)$ are the standard deviation and mean of all pairwise distances,
- $\text{clip}(CV, CV_{\min}, CV_{\max})$ clips CV into the range $[CV_{\min}, CV_{\max}]$,
- $Q_{\pi(CV)}(D)$ returns the $\pi(CV)$-th percentile of the distance matrix D,
- τ_{cluster} is the final adaptive clustering threshold.

We employ the following parameter values: $\pi_{\max} = 90$, $\pi_{\min} = 60$, $\pi_{\text{range}} = 30$, $CV_{\min} = 0.1$, and $CV_{\max} = 1.0$. These values ensure that the percentile threshold ranges from the 60th percentile (for homogeneous data with low CV) to the 90th percentile (for heterogeneous data with high CV), adapting the clustering threshold to dataset-specific variability. This approach produces compact, internally similar clusters while avoiding over-fragmentation when data exhibits high variance.

6.3.5 Test Selection

Once clusters are defined, we identify representative sections from each cluster to achieve our selection objective: *minimize the number of test cases while ensuring comprehensive coverage of all geometric and behavioral patterns present in the original dataset.* By selecting representatives from each cluster, we maintain diversity across the full spectrum

of road characteristics while eliminating redundancy by avoiding the execution of multiple nearly-identical roads that would provide minimal additional validation value.

We distinguish two cases: singleton clusters and non-singleton clusters. For each singleton cluster $C = \{s\}$, we select as representative the only section included in C, that is $Rep(C) = s$. We refer to the set of all the representatives collected from singleton clusters with $Rep_{singleton}$. For each cluster with multiple sections, we use a diversity-driven framework to select up to three representatives per cluster. The rationale for three representatives is to balance coverage granularity with computational efficiency: a single representative per cluster risks oversimplifying the cluster's diversity (all sections within a cluster may still exhibit meaningful behavioral variation), while selecting all sections defeats the purpose of clustering-based reduction. Three representatives capturing low, medium, and high curvature variants within a cluster provide a reasonable trade-off, ensuring that different intensities of driving challenge within each cluster are represented without excessive test proliferation. Specifically, if the cluster includes three or fewer sections, we select all the sections. If a cluster contains more than three sections, we compute the mean curvature of each section, and we select three representatives equally distributed across the spectrum of curvature values, intuitively selecting representatives with low, medium, and high curvature. This ensures that clusters spanning a range of geometric complexity are adequately represented in the final test suite.

The curvature $\bar{\kappa}(S)$ of a section S is computed as follows:

$$\bar{\kappa}(S) = \frac{1}{|S|} \sum_{i=1}^{|S|} \kappa_i \qquad (6.11)$$

where: κ_i = curvature at point i of section S, $|S|$ = number of points in section S, and $\bar{\kappa}(S)$ = mean curvature of section S.

We indicate this set of representative sections as R_{multi}. The selected test cases aim to first cover the representative sections, that is, $REP = R_{singleton} \cup R_{multi}$. Once REP has been identified, we select a subset of tests $T_{cov} \subseteq T$ that covers all representative sections. Because multiple tests may contain the same representative section, we adopt a greedy strategy inspired by the classical set cover heuristic: at each step, we choose the test that covers the largest number of currently uncovered representative sections [275].

Formally, we initialize $T_{cov} \leftarrow \emptyset$ and $REP_{rem} \leftarrow REP$. While $REP_{rem} \neq \emptyset$, we select a test:

$$T^{\star} = \arg\max_{T_i \in T} |REP_{rem} \cap Sections(T_i)|$$

where $Sections(T_i)$ denotes the set of road sections (segments) contained in test T_i. We then update $T_{cov} \leftarrow T_{cov} \cup \{T^{\star}\}$ and $REP_{rem} \leftarrow REP_{rem} \setminus Sections(T^{\star})$.

The output of test selection therefore splits the initial test suite T into two test suites T_{cov}, which include the high-priority test cases, and $T_{surplus} = T \setminus T_{cov}$, which includes the remaining low-priority test cases.

6.3.6 Test Prioritization

Once T_{cov} and $T_{surplus}$ have been defined, we determine the execution order of the tests inside each group using a multi-criteria prioritization framework that combines geometric complexity, dynamic difficulty, and historical performance. That is, the tests that use roads with the most complex shape, produce the most challenging ADAS behavior, and have already failed in past executions are executed first. To achieve this result, we introduce a test scoring mechanism based on the combination of three scores.

Geometric score: The geometric complexity score captures road shape diversity using the metrics curvature variation, number of high-curvature sections, and diversity of section types (left/right/straight). For each test $T = (R, C)$:

$$G(T) = w_{cv} \cdot \sigma(curv) + w_{hc} \cdot N_{hc} + w_{dt} \cdot D_{types} \tag{6.12}$$

where: $\sigma(curv)$ is the standard deviation of curvature values (curvature variability), N_{hc} is the number of high-curvature sections in R exceeding the threshold $\kappa_{thr} = 0.015$ rad/m (corresponding to curves with radius less than 66.67 meters), and D_{types} is the number of distinct section types among {straight, left, right})($D_{types} \in \{1, 2, 3\}$), with $w_{cv} = w_{hc} = w_{dt} = \frac{1}{3}$ for balanced geometric diversity assessment. While geometric variety challenges autonomous systems by requiring adaptation to changing control demands [276], section type diversity ensures that tests exercise multiple aspects of control capability, such as straight-line tracking, left-turn navigation, and right-turn handling, rather than repeatedly evaluating the same behaviors. Roads with higher type diversity, therefore, provide more comprehensive validation per test execution [178].

Dynamic score Dynamic difficulty scoring $D(T)$ quantifies scenario difficulty using behavioral metrics derived from vehicle execution, namely speed variability, steering variability, cross-track error (CTE) severity, and yaw rate variability.

Each test case $T = (R, C)$ consists of a road R that is partitioned into contiguous road segments, as described in Section 6.3.1.3. During execution, we collect time-series driving data for each segment and compute segment-level indicators: the standard deviation of vehicle speed σ_v, the standard deviation of the steering angle σ_θ, the mean absolute cross-track error cte, and the standard deviation of the yaw rate σ_ψ.

To obtain a test-level difficulty score, we aggregate segment-level indicators using a

length-weighted mean, ensuring that longer and more influential road segments contribute proportionally to the overall score. Formally, each segment-level feature is first aggregated across segments, then min–max normalized to $[0, 1]$ at the test level. The final dynamic difficulty score is computed as an equal-weighted average of the normalized indicators:

$$D(T) = \frac{1}{4}\left(\sigma_v + \sigma_\theta + \text{cte} + \sigma_\psi\right) \tag{6.13}$$

This score captures the empirical difficulty the autonomous system experienced while navigating the entire road. High values indicate scenarios that require frequent speed adjustments, substantial steering corrections, significant deviation from the reference trajectory, or pronounced directional instability. Prior work has shown that such behavioral indicators correlate with proximity to failure boundaries and increased failure likelihood [19, 77].

Historical performance score: Historical performance score $H(T)$ incorporates evidence from previous test executions to identify roads that previously exposed failures. For regression testing scenarios where the test suite has been executed with earlier system versions, historical data provides valuable information about which roads consistently challenge the system or reveal newly introduced bugs [79, 277]. The historical score follows a simple binary scheme:

$$H(T) = \begin{cases} 0.25 & \text{if } T \text{ failed in previous executions} \\ 0 & \text{otherwise} \end{cases}$$

The constant 0.25 was chosen to provide a meaningful but not dominant influence; historical evidence should elevate priority, but not override geometric and dynamic scores entirely. This balance acknowledges that while past failures predict future failures, systems evolve, and previously failing tests might pass after bug fixes, while previously passing tests might reveal new issues [278]. The limited contribution ensures that prioritization adapts to evolving system characteristics rather than becoming locked to historical patterns.

The final priority score combines geometric, dynamic, and historical components:

$$P(r) = \alpha \cdot G(T) + \beta \cdot D(T) + H(T) \tag{6.14}$$

where $\alpha + \beta = 1$, with default $\alpha = 0.5$ (geometric emphasis) and $\beta = 0.5$ (dynamic emphasis). All scores are normalized to $[0, 1]$ before weighting to ensure comparability.

The final execution order inside each group of tests is determined by descending values of $P(r)$. As a result, we get a ranked list of roads, each with a transparent breakdown of

its contributing factors and rationale for its assigned priority. This ensures that the most challenging and historically problematic scenarios are executed first, supporting efficient resource allocation and maximizing fault-detection value.

Our multi-criteria prioritization reflects complementary factors known to influence ADAS failures. Geometric complexity is captured through curvature variation and high-curvature segments, as small-radius and rapidly changing curves are empirically linked to control instability; the curvature threshold $\tau_c = 0.015m^{-1}$ follows geometric design guidance and prior ADAS studies [279, 235]. While geometry captures structural difficulty, dynamic metrics (e.g., steering variability, cross-track error, yaw, and speed variation) capture runtime difficulty beyond geometry and have been shown to be effective for characterizing misbehavior-inducing scenarios [280, 281]. Finally, historical failure information follows established regression-testing principles, where previously failing or similar tests are more likely to fail again [79, 278]. Equal weighting of geometric and behavioral scores $\alpha, \beta = 0.5$ reflects their comparable predictive value but can be adjusted when behavioral transferability is limited across architectures.

The framework assumes that geometric and behavioral features provide complementary signals and that past failures and extreme behaviors are predictive of future risk; however, these signals are often correlated (geometrically complex roads induce challenging behaviors), model and scenario dependent, and unevenly reliable, which can bias prioritization under unseen architectures or environments, highlighting an inherent trade-off between adaptability and generality [282]. Despite these limitations, our empirical evaluation in section 6.4 shows that the multi-criteria framework consistently outperforms single-criterion baselines (random and geometry-only) across all campaigns, the selected suite achieves an average reduction of 89% while retaining 75% of failures (see Table 6.2 and Table 6.3), indicating that the proposed approach yields more reliable prioritization than relying on any single criterion alone.

Figure 6.2 provides a concrete illustration of the proposed framework, demonstrating the sequential operations described in Sections 6.3.1-6.3.6.

In Part *(a)*, the input set of roads $R = \{R_1, R_2, R_3, R_4\}$ is decomposed into multiple sections $S = \{S_1, S_2, \ldots, S_{12}\}$ through curvature-based segmentation. This process captures local geometric variations such as straight, left-curved, and right-curved sections, enabling fine-grained road comparison.

Part *(b)* illustrates the section matching and clustering phase. Each section is compared against others based on its section type (straight vs. straight and so on), leading to the formation of clusters $C = \{C_1, C_2, C_3\}$. Each cluster groups similar sections and is represented by a *Cluster Representative* (CR) (as described in subsection 6.3.5), highlighted by a red dot in the figure. The CR serves as the most informative example of that section

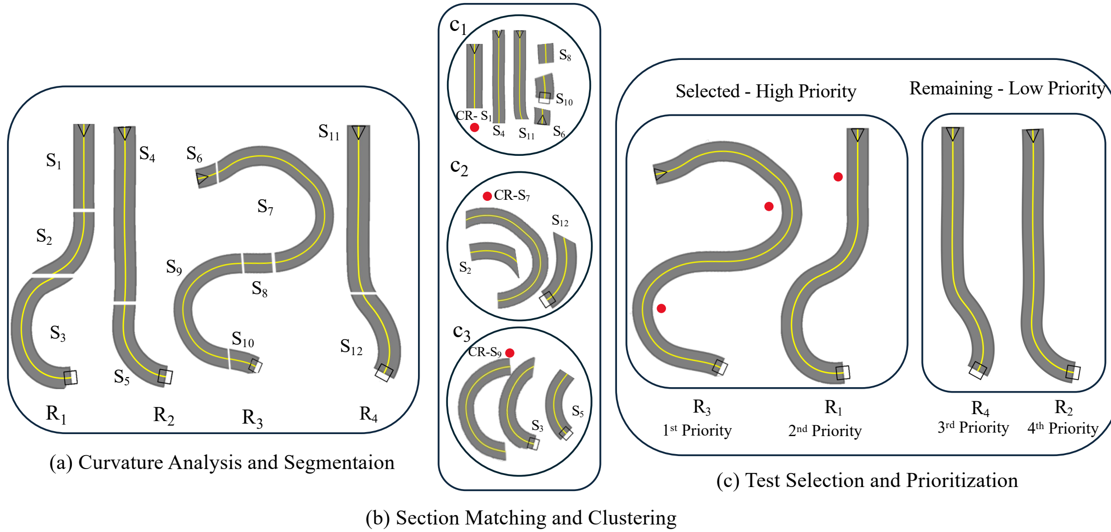


Figure 6.2: Illustrative example of the proposed framework

type (e.g., straight, a sharp curve, or a complex curvature pattern).

Finally, Part *(c)* presents the test selection and prioritization process. Roads containing one or more cluster representatives (e.g., R_3 and R_1) are selected to form the minimal yet diverse test suite, ensuring that the selected set covers the distinct geometric and dynamic behaviors identified across the dataset. These roads are assigned higher priority for testing (e.g., R_3 first and R_1 second), while the remaining roads (R_4 and R_2) are retained as lower priority candidates, providing supplementary coverage if additional testing resources permit.

This example illustrates how the proposed framework systematically decomposes complex road networks into comparable structural units, clusters them to identify representative geometric and dynamic behaviors, and subsequently prioritizes test scenarios based on their diversity and criticality.

6.4 Empirical Evaluation

The empirical evaluation assesses the framework's effectiveness in reducing test suite size and improving overall testing efficiency. Specifically, we assess how well the combination of geometric segmentation, dynamic data integration, and historical failure information reduces and prioritizes large-scale ADS test suites while preserving fault detection capability (RQ_3, Chapter 1).

6.4.1 Research Questions

The evaluation addresses four sub-questions derived from RQ_3, each examining a different aspect of the prioritization framework's effectiveness.

RQ_{3a} (Selection): *How effective is the selected test suite?*

This research question investigates the cost-effectiveness of the test suite without proportionally sacrificing fault detection capability. A good test suite must discover a good number of issues using a few tests only.

RQ_{3b} (Effectiveness of Dynamic Data): *How does combining dynamic driving behavior with road geometry affect test selection quality and fault detection compared to using either feature type alone?*

This research question examines whether combining dynamic driving behavior with road geometry improves test selection and fault detection compared to using geometric or dynamic features in isolation.

RQ_{3c} (Prioritization): *How effective is the proposed prioritization strategy in identifying critical scenarios and detecting failures?*

This research question investigates whether the proposed prioritization method can schedule test executions in a way that critical scenarios are likely to be executed early, compared to a random order of the tests.

RQ_{3d} (Model Transferability): *To what extent do test selection and prioritization strategies transfer across architecturally different ADAS models.*

This research question investigates whether roads that induce failures in one ADAS architecture also induce failures in a fundamentally different architecture, and how using model-specific dynamic data affects failure detection capability across models. Importantly, it characterizes the empirical *boundaries* of transferability between two architecturally distinct end-to-end driving models rather than universal generalization.

6.4.2 Objects of Study

To assess test selection and prioritization, we consider test suites exercising NHTSA [283] Level 2 ADAS, which perform vision-based perception from on-board cameras and provide partial driving automation (e.g., lateral control and optionally longitudinal control) while still requiring human supervision. Despite their widespread deployment, the safety and reliability of Level 2 systems remain a concern, as documented by various recent crash reports [284] and real-world validation experiments [285]. Although higher levels of automation (Levels 3-4) have been proposed [286], their real-world deployment remains highly constrained. Consequently, addressing the limitations of Level 2 systems is crucial for advancing to higher levels of autonomy.

We focus on a specific ADAS application, that is, a system for lane-keeping assistance (LKA). LKA is an ADAS that uses cameras and sensors to detect lane markings, providing steering corrections and/or warnings to help prevent unintentional drifting out of a lane. From a technical perspective, the lane-keeping task is implemented using behavior

cloning–based deep neural networks (DNNs), which learn end-to-end control policies from supervised expert demonstrations. The training data consists of driving images captured by an onboard camera sensor and paired with corresponding control commands provided by a human driver[120].

In this work, we consider lane-keeping models such as DAVE-2 and Chauffeur, which predict the steering angle required to keep the vehicle within lane boundaries given a single input image. DAVE-2 is a convolutional neural network developed for multi-output regression tasks based on imitation learning [52]. The model architecture includes three convolutional layers for feature extraction, followed by five fully connected layers. Chauffeur[53] is an alternative end-to-end lane-keeping architecture that differs from DAVE-2 by incorporating temporal modeling capabilities. Chauffeur combines convolutional feature extraction with recurrent Long Short-Term Memory (LSTM) layers to leverage temporal dependencies across sequential frames, which can lead to smoother control but also to different failure modes compared to frame-based models. This architectural diversity allows assessing whether the same selected/prioritized scenarios remain effective across distinct LKA model families (CNN vs. CNN+LSTM) [134].

Both models have been extensively used in a variety of ADAS testing studies [28, 287, 288, 120, 261, 262]. These models are typically trained on stationary datasets using stochastic gradient descent [289], with the objective of minimizing the error between predicted and ground-truth steering commands. Control labels generally include steering, throttle, and brake commands; however, in the simplest configuration, only the steering angle is provided, while the throttle is computed as a function of steering input and vehicle speed.

6.4.3 Experimental Platforms and Benchmarks

We conducted experiments on the Udacity Self-Driving Car Simulator (Term 3, v1.2) [290] since it is open-source and suitable for Level 2 ADAS evaluation. Simulation platforms are widely used for testing of ADAS, as researchers have shown that model-level testing is inadequate at exposing system-level failures [291, 263, 292]. Udacity [290] is developed with Unity 3D [113] and distributed as pre-compiled binaries. Unity is a popular cross-platform game engine, based on the Nvidia PhysX engine [114], featuring discrete and continuous collision detection, ray-casting, and rigid-body dynamics simulation.

As a benchmark, we used the test scenarios available in the OpenCat dataset [149], which provides 32,580 different road scenarios converted from the SensoDat benchmark [40] across three campaigns: *Ambiegen, Frenetic, and Frenetic_v*, each generated by a different BeamNG.tech test generator and reflecting distinct testing philosophies. **Ambiegen** [151] applies a multi-objective NSGA-II approach [293], optimizing diversity and fault-revealing capability to generate OOB-focused test cases, and **Frenetic** [294] uses a genetic algorithm

to minimize the distance between the Self-Driving Car (SDC) and the road edge, operating in frenet-frame, curvature-based representations that are converted back to Cartesian space for simulation, while **Frenetic_v** [295] extends Frenetic by reducing invalid roads (e.g., sharp turns or self-intersections), following the SBST Tool Competition road validity definition [296].

6.4.4 Procedure and Metrics

The experimental procedure evaluates the proposed framework by examining its ability to select road scenarios that provide broad coverage of geometric and behavioral patterns while minimizing redundancy ($\mathrm{RQ_{3a}}$, $\mathrm{RQ_{3b}}$), by assessing how effectively the resulting test suites are prioritized to expose critical or failure-prone scenarios early in the testing process ($\mathrm{RQ_{3c}}$), and by evaluating cross-model transferability through selection and prioritization performed using one model's behavioral data and execution of the resulting test suite on a different model ($\mathrm{RQ_{3d}}$). The evaluation metrics used to address each research question are described in detail below.

6.4.4.1 $\mathbf{RQ_{3a}}$ (Selection)

We evaluate the effectiveness of coverage-based test selection in comparison to random test ordering using the following metrics:

Reduction Ratio (%): The reduction ratio quantifies the efficiency of the test selection process by measuring the percentage of tests eliminated from the original test suite. A higher ratio indicates more efficient reduction. It is defined as:

$$\text{Reduction Ratio (\%)} = \frac{|T_{\text{surplus}}|}{|T|} \times 100 \tag{6.15}$$

Failed Roads Retention (%): This metric measures the effectiveness of the reduction by considering the percentage of failed tests in the selection. It is defined as:

$$\text{Failed Roads Retention (\%)} = \frac{|T_{cov} \cap F|}{|F|} \times 100 \tag{6.16}$$

where F represents the set of failed tests, and T_{cov} is the selected set. This metric validates whether the selection process retains challenging geometries previously associated with test failures, ensuring that critical test cases remain part of the reduced test suite. We consider the average number of failed roads discovered by a random selection of N tests, for multiple values of N, as a baseline.

6.4.4.2 $\mathbf{RQ_{3b}}$ (Effectiveness of Dynamic Data)

To address this question, we compare three configurations of the test selection pipeline: *Geometric-only selection (Geo-only)*, which relies exclusively on road geometry features as described in subsection 6.3.2; *Dynamic-only selection (Dynamic-only)*, which uses only dynamic driving behavior features as described in subsection 6.3.3; and a *Hybrid configuration (Geo + Dynamic)*, which integrates both feature types.

All three configurations are applied to the same set of road scenarios across all campaigns to enable a controlled comparison. Their effectiveness is evaluated based on the proportion of historically failed roads retained within the reduced test suite, using the same selection and prioritization metrics adopted in $\mathrm{RQ_{3a}}$ and $\mathrm{RQ_{3c}}$.

6.4.4.3 $\mathbf{RQ_{3c}}$ (Prioritization)

We use the following metrics to evaluate the performance of the prioritization approach.

Early Fault Detection (EFD): This metric measures the percentage of failed tests selected among the first k tests in the prioritized list. We use this metric to evaluate early fault detection capability, quantifying how quickly our approach identifies safety-critical scenarios compared to random ordering. We consider two cases, $k = 10$ to assess the capability to observe misbehaviors immediately after the test suite is executed, and $k = |T_{cov}|$ (which is the same number of tests selected by our approach), to assess how good the initial part of the test suite is. When considering random ordering, we compute the actual average percentage of failed tests that occur in the first k tests.

APFD (Average Percentage of Fault Detection) Calculation: We compute how good a test ordering is using the well-established APFD [297] metric:

$$APFD = 1 - \frac{\sum_{i=1}^{m} TF_i}{n \times m} + \frac{1}{2n}$$

where TF_i represents the position of the first test revealing fault i, n is the total number of tests (roads), and m is the total number of failures. Higher APFD values indicate better fault detection across the test execution sequence.

6.4.4.4 $\mathbf{RQ_{3d}}$ (Model Transferability)

We evaluate transferability between DAVE-2 and Chauffeur [53] across 36 campaigns spanning Ambiegen, Frenetic, and Frenetic_v. The two models represent architecturally distinct end-to-end controllers: DAVE-2 is a feedforward network with five convolutional and three fully connected layers [52], whereas Chauffeur uses a deeper convolutional front-end and integrates recurrent LSTM layers to exploit temporal context. This architectural

diversity allows us to assess whether road suites derived from one model's behavioral profile remain effective when testing a different model and, conversely, to what extent failures are model-specific rather than scenario-intrinsic.

For each target model, we compare (i) an *own-profile* configuration, where dynamic features are extracted from that same model's traces, against (ii) a *cross-profile* configuration, where the dynamic features used in clustering are taken from the other model, while failures are always measured on the target model. Concretely, we report DAVE-2 failures retained by selection when clustering uses either DAVE-2 traces (own-profile) or Chauffeur traces (cross-profile), and symmetrically Chauffeur failures retained when clustering uses either Chauffeur traces (own-profile) or DAVE-2 traces (cross-profile).

This design lets us answer two complementary transfer questions. First, we assess *selection stability* by comparing how much of the campaign suite is selected when clustering is driven by DAVE-2 vs. Chauffeur behavioral profiles, which captures whether changing the source model meaningfully alters the reduced suite size. Second, we assess *cross-profile effectiveness* by quantifying the degradation in failure retention when the behavioral difficulty signal is sourced from a different architecture, and we complement failure-retention results with overlap measures (Jaccard similarity and conditional failure overlap) to separate shared vs. model-specific failure patterns.

Jaccard Similarity (Symmetric Failure Overlap): The Jaccard similarity coefficient [298], also known as the Jaccard index, measures the proportion of shared failures among all roads failed by at least one model. In our context, it captures the extent to which failure-inducing road characteristics are architecture-agnostic.

Let F_D and F_C denote the sets of roads on which DAVE-2 and Chauffeur fail, respectively, as reported in Table 6.4 (columns 6 and 9). The *Jaccard similarity coefficient* is defined as:

$$J(F_D, F_C) = \frac{|F_D \cap F_C|}{|F_D \cup F_C|} = \frac{|F_D \cap F_C|}{|F_D| + |F_C| - |F_D \cap F_C|} \tag{6.17}$$

Where $|F_D \cap F_C|$ represents the number of roads that failed in both models (overlapping failures), and $|F_D \cup F_C|$ represents the total number of unique roads that failed in at least one model. The Jaccard coefficient ranges from 0 (completely disjoint failure sets) to 1 (identical failure sets).

Conditional Failure Overlap (Directional Transferability): While Jaccard similarity captures symmetric overlap, it does not reveal directional dependencies between failure sets. To quantify directional transferability, we introduce conditional failure overlap based on empirical conditional probability:

$$P(F_C \mid F_D) = \frac{|F_D \cap F_C|}{|F_D|} \tag{6.18}$$

and conversely,

$$P(F_D \mid F_C) = \frac{|F_D \cap F_C|}{|F_C|} \tag{6.19}$$

where $P(F_C \mid F_D)$ represents the conditional probability that Chauffeur fails on a road given that DAVE-2 has failed on the same road (and vice versa, $P(F_D \mid F_C)$). This metric ranges from 0 (no directional overlap) to 1 (perfect directional overlap). Asymmetric values of $P(F_C \mid F_D)$ and $P(F_D \mid F_C)$ indicate directional dependencies in failure patterns.

The conditional metric complements Jaccard similarity by revealing asymmetries relevant to practical decisions: if $P(F_C \mid F_D) > P(F_D \mid F_C)$, then the DAVE-2-based test suite provides better coverage for Chauffeur failures than the reverse, even when symmetric overlap appears moderate. This directional information is critical for multi-model testing strategies where practitioners must decide which model's behavioral profile to use as the basis for a shared test suite or whether model-specific profiling is required.

6.4.5 Results

Table 6.2 and Table 6.3 present comprehensive results for coverage-based road selection and prioritization experiments conducted on a total of 32,580 test roads across the three OpenCat [149] campaigns: *Ambiegen*, *Frenetic*, and *Frenetic_v*. Columns *Campaign*, *total No. Tests* and *No. Failed Tests* indicate the ID of the campaign, the number of tests available in that campaign, and the total number of tests failed by DAVE-2 and Chauffeur in the campaign, respectively. Columns *Selected* and *Reduction* indicate the absolute number and percentage of tests selected by our approach (i.e., the tests in $|T_{cov}|$), respectively. Column *FRR Selected %* reports the percentage of failures that are revealed by the selected test cases. Column *EFD RnD* indicates the average percentage of faults discovered by randomly selecting as many tests as the ones in $|T_{cov}|$. Columns *EFD10 Tests* and *EFD10 Rnd Tests* show the percentage of failures revealed by the first 10 test cases selected with our approach and randomly, respectively. Finally, column *APFD* indicates the Average Percentage of Fault Detection metric for the prioritized test suite.

6.4.5.1 RQ$_{3a}$ (Selection)

Our approach substantially reduces test suite size while preserving representative geometric and behavioral diversity. Across all campaigns, it achieves an average reduction of 89% for DAVE-2 and 88% for Chauffeur, while retaining 73% and 75% of faults respectively. These results indicate that the selection mechanism effectively filters redundant roads while preserving those most likely to expose failures, and that this behavior generalizes

Table 6.2: Coverage-based selection and prioritization-DAVE-2.

Campaign	Total No. Tests	No. Failed Tests	Selected Tests	Reduction %	FRR Selected%	EFD RnD %	EFD10 Tests	EFD10 Rnd Tests	APFD
Ambiegen Campaigns									
2	973	11	147	85%	45%	0.17%	45%	1.04%	0.92
3	964	9	206	79%	89%	0.20%	80%	1.04%	0.95
4	965	5	178	82%	80%	0.10%	80%	1.04%	0.93
5	958	10	167	83%	80%	0.18%	70%	1.04%	0.91
6	959	9	179	81%	78%	0.18%	70%	1.04%	0.89
7	963	10	197	80%	70%	0.21%	60%	1.04%	0.96
8	952	11	176	82%	91%	0.21%	91%	1.05%	0.92
9	953	4	187	80%	100%	0.08%	75%	1.05%	0.97
10	971	18	176	82%	89%	0.34%	56%	1.03%	0.85
11	973	10	190	80%	80%	0.20%	80%	1.03%	0.93
13	954	7	185	81%	86%	0.14%	86%	1.05%	0.94
14	959	8	187	80%	75%	0.16%	75%	1.05%	0.82
15	952	19	206	78%	63%	0.43%	53%	1.05%	0.96
Frenetic Campaigns									
2	928	7	27	97%	57%	0.02%	57%	1.08%	0.88
3	954	11	41	96%	73%	0.05%	73%	1.06%	0.91
4	964	12	29	97%	58%	0.04%	58%	1.05%	0.92
5	945	8	30	97%	63%	0.03%	75%	1.06%	0.93
6	944	16	33	97%	44%	0.06%	44%	1.06%	0.90
7	967	14	38	96%	57%	0.03%	57%	1.03%	0.97
8	952	10	30	97%	60%	0.03%	70%	1.05%	0.86
9	964	6	37	97%	67%	0.04%	67%	1.04%	0.97
11	866	11	37	96%	64%	0.05%	64%	1.15%	0.94
12	956	17	39	96%	59%	0.07%	59%	1.05%	0.89
13	959	13	34	96%	54%	0.05%	54%	1.04%	0.95
14	866	11	33	96%	73%	0.05%	73%	1.15%	0.97
15	870	12	37	96%	67%	0.06%	67%	1.15%	0.85
Frenetic_v Campaigns									
2	944	7	31	97%	86%	0.02%	87%	1.06%	0.92
4	525	3	25	95%	67%	0.03%	67%	1.90%	0.87
5	940	7	21	98%	100%	0.02%	100%	1.06%	0.95
6	764	5	22	97%	100%	0.02%	100%	1.31%	0.92
7	47	0	8	83%	-	-	-	-	-
11	953	8	33	97%	88%	0.03%	88%	1.05%	0.90
12	942	7	27	97%	71%	0.02%	71%	1.06%	0.87
13	951	13	34	96%	54%	0.03%	87%	1.05%	0.83
14	934	9	35	96%	89%	0.04%	89%	1.07%	0.90
15	949	7	33	97%	86%	0.03%	86%	1.05%	0.85

across architecturally distinct ADAS models.

Ambiegen Campaigns: In the Ambiegen campaigns, the framework reduces road sets by an average of 81% for both models while retaining a large proportion of failures (mean 79% for DAVE-2 and 77% for Chauffeur). Importantly, selection sizes remain highly consistent (approximately 19%) despite Chauffeur exhibiting a higher total number of failures. This indicates that the selection process is driven primarily by scenario characteristics rather than model-specific failure counts, while still preserving comparable fault exposure for both architectures.

Frenetic Campaigns: For Frenetic campaigns, the framework produces even higher reductions (an average of 96%) while maintaining moderate fault retention (mean 64% for

Table 6.3: Coverage-based selection and prioritization-Chauffeur.

Campaign	Total No. Tests	No. Failed Tests	Selected Tests	Reduction %	FRR Sel.%	EFD RnD %	EFD10 Tests	EFD10 Rnd Tests	APFD
Ambiegen Campaigns									
2	973	16	151	85%	44%	0.16%	44%	1.03%	0.93
3	964	11	208	78%	91%	0.11%	91%	1.04%	0.96
4	965	7	180	81%	86%	0.07%	86%	1.04%	0.92
5	958	10	173	82%	80%	0.11%	70%	1.04%	0.91
6	959	13	183	81%	62%	0.14%	62%	1.04%	0.86
7	963	9	193	80%	45%	0.09%	45%	1.04%	0.90
8	952	14	179	81%	72%	0.15%	65%	1.05%	0.93
9	953	7	185	81%	100%	0.07%	100%	1.05%	0.97
10	971	12	189	81%	83%	0.13%	83%	1.03%	0.92
11	973	10	189	81%	90%	0.10%	90%	1.03%	0.94
13	954	11	189	81%	100%	0.12%	86%	1.05%	0.95
14	959	9	185	81%	89%	0.09%	89%	1.05%	0.91
15	952	16	151	84%	75%	0.17%	63%	1.05%	0.82
Frenetic Campaigns									
2	928	7	32	97%	57%	0.08%	57%	1.06%	0.88
3	954	13	48	95%	62%	0.14%	62%	1.05%	0.88
4	964	13	42	96%	85%	0.14%	77%	1.04%	0.91
5	945	14	49	95%	71%	0.15%	71%	1.06%	0.90
6	944	16	38	96%	44%	0.17%	44%	1.06%	0.88
7	967	15	23	98%	40%	0.16%	40%	1.03%	0.93
8	952	11	33	97%	55%	0.12%	55%	1.05%	0.92
9	964	10	39	96%	80%	0.10%	80%	1.04%	0.94
11	866	11	38	96%	81%	0.13%	81%	1.15%	0.95
12	956	16	39	96%	56%	0.15%	56%	1.04%	0.88
13	959	15	35	96%	67%	0.16%	67%	1.04%	0.91
14	866	9	34	96%	56%	0.10%	56%	1.15%	0.87
15	870	9	36	96%	44%	0.10%	44%	1.15%	0.84
Frenetic_v Campaigns									
2	944	8	37	96%	100%	0.08%	100%	1.06%	0.95
4	525	4	33	94%	75%	0.08%	75%	1.90%	0.91
5	940	6	35	96%	100%	0.06%	100%	1.06%	0.97
6	764	6	26	97%	83%	0.08%	83%	1.31%	0.92
7	47	0	9	89%	-	-	-	-	-
11	953	8	31	97%	75%	0.08%	75%	1.05%	0.90
12	942	14	33	97%	93%	0.15%	72%	1.06%	0.96
13	951	10	37	96%	90%	0.10%	90%	1.05%	0.93
14	934	8	30	97%	100%	0.09%	100%	1.07%	0.95
15	949	6	29	97%	67%	0.06%	67%	1.05%	0.89

both models). At the same time, selection sizes remain stable (approximately 3.6-4.0%), and total failure counts converge across models. Together, these trends suggest that extreme geometric difficulty leads to largely architecture-independent selection behavior, with similar subsets of roads deemed critical for both models.

Frenetic_v Campaigns: This pattern continues For Frenetic_v campaigns, where the framework again achieves an average of 96% reduction but with higher fault retention (mean 85% for DAVE-2 and 88% for Chauffeur). Selection sizes differ by only 0.3 percentage points (DAVE-2 selecting 3.3% and Chauffeur 3.6%), and multiple campaigns achieve full fault retention, indicating that when failures are concentrated in well-defined critical

scenarios, the selection mechanism captures them particularly effectively.

Cross-Model Analysis: Across all campaign types, large reductions (81-96%) are accompanied by remarkably similar selection sizes for Dave-2 (feedforward) and Chauffeur (recurrent) models. This consistency indicates that geometric features dominate the clustering process, while behavioral dynamics refine which specific failures are preserved. Consequently, differences in retained failures reflect model-specific sensitivities rather than instability in the selection mechanism. Overall, although Chauffeur exhibits slightly higher fault retention, this occurs within nearly identical selected subsets, suggesting that its failures are more densely concentrated in critical road segments.

These results demonstrate that the proposed selection mechanism reliably enables early discovery of safety-critical failures while allowing remaining tests to be executed as resources permit, and that it adapts robustly across different ADAS control architectures.

6.4.5.2 RQ_{3b} (Effectiveness of Dynamic Data)

Figure 6.3 and Figure 6.4 compare test selection size and fault detection effectiveness for DAVE-2 and Chauffeur under three feature configurations: geometric-only, dynamic-only, and their hybrid combination. This comparison isolates the individual and combined contributions of road geometry and dynamic behavioral information.

Across all campaign types and both models, the hybrid configuration consistently outperforms either feature set used in isolation. While geometric-only and dynamic-only features each identify a limited subset of failure-prone roads (typically 3-25% FRR), their combination yields substantial gains, improving fault detection by 60-83 percentage points depending on campaign type and architecture. This indicates a strong synergistic effect rather than an additive contribution.

DAVE-2: For the feedforward model, hybrid selection achieves high fault detection across all campaigns, reaching FRR of 82% in Ambiegen, 70% in Frenetic, and 65% in Frenetic_v scenarios. In contrast, geometric-only and dynamic-only configurations remain below 22% across all cases. Although incorporating dynamic data increases selection size by approximately 9%, the resulting improvement in fault detection far outweighs this modest expansion, indicating that behavioral features meaningfully refine geometric clustering by distinguishing which structurally similar roads actually induce failures.

Chauffeur: Similar trends hold for the recurrent model, with hybrid fault detection reaching FRR of 77% (Ambiegen), 64% (Frenetic), and 88% (Frenetic_v). Notably, Chauffeur consistently benefits more from dynamic-only features than DAVE-2, reflecting the sensitivity of recurrent architectures to temporal control patterns. The particularly strong hybrid performance in Frenetic_v campaigns suggests that when scenarios are

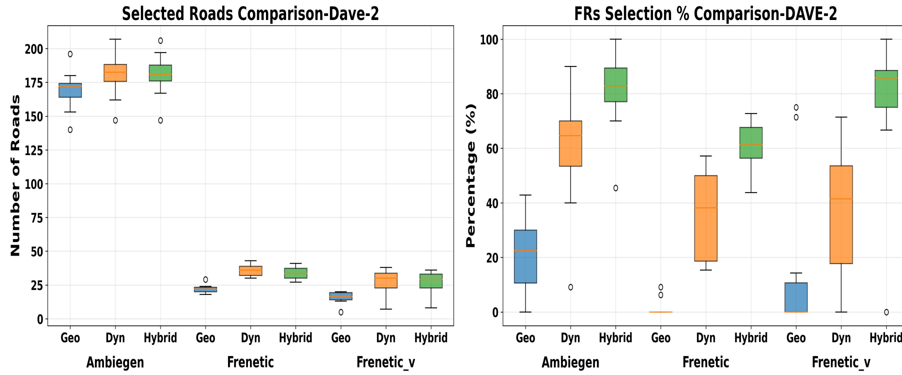


Figure 6.3: Comparison between geometric-only and hybrid approaches (Dave-2).

curated to induce failures, temporal behavioral signatures become especially diagnostic for LSTM-based control.

Cross-Model Comparison: Comparing feature effectiveness across models reveals clear architectural sensitivities. Chauffeur slightly outperforms DAVE-2 under both geometric-only and dynamic-only configurations, indicating that recurrent architectures align more closely with individual feature signals. However, hybrid performance exhibits campaign-dependent reversals: DAVE-2 performs better in Ambiegen and Frenetic scenarios, while Chauffeur substantially outperforms DAVE-2 in Frenetic_v campaigns. This pattern suggests that geometric-behavioral synergy depends on how failures arise—instantaneous geometric difficulty versus accumulated temporal control error.

Implications for Framework Design: These results confirm that dynamic behavioral data is essential for effective test selection, but insufficient on its own. Road geometry provides the primary structural grouping, while behavioral features determine which specific roads within similar geometric clusters are failure-inducing for a given architecture. The consistent superiority of the hybrid approach validates the framework's design choice to integrate both feature types. At the same time, the observed architectural differences suggest that feature weighting may benefit from model-specific tuning: balanced weighting is effective for feedforward models, while recurrent models may benefit from higher emphasis on dynamic features in validated critical scenarios.

Overall, RQ_{3b} demonstrates that effective test selection for autonomous driving requires coupling geometric complexity with dynamic behavioral responses, and that this coupling is both necessary and sufficient to achieve substantial gains in fault detection across diverse ADAS architectures.

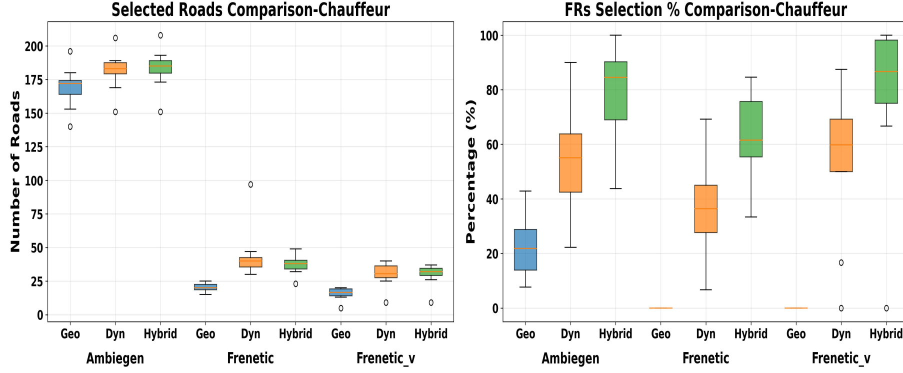


Figure 6.4: Comparison between geometric-only and hybrid approaches (Chauffeur).

6.4.5.3 RQ_{3c} (Prioritization)

To evaluate whether selected tests are executed in an effective order, Table 6.2 and Table 6.3 report Early Fault Detection (EFD) for prioritized test suites compared to random selection, including the $EFD-Top10$ metric assessing the effectiveness of the first 10 tests, studying how effective the very top selected test cases are in revealing potential issues in the ADS under test.

Across all campaigns, the hybrid prioritization achieves approximately 60-95× higher early fault detection than random. This consistent improvement confirms that the ranking mechanism successfully elevates safety-critical roads early in the execution order for both DAVE-2 and Chauffeur.

Ambiegen Campaigns: In Ambiegen driving scenarios, $EFD-Top10$ ranges from 53-91% for DAVE-2 (mean 72%) and 44-100% for Chauffeur (mean 74%), compared to a random baseline near 1%. Although campaign-level variations exist, mean performance is slightly higher for Chauffeur, indicating that the prioritization mechanism remains effective even when the recurrent model exhibits higher overall fault counts. At the same time, several campaigns show near-identical performance across models, suggesting largely architecture-agnostic prioritization under naturalistic conditions.

Frenetic Campaigns: In Frenetic scenarios, prioritization performance converges further, with mean $EFD-Top10$ of 62% for DAVE-2 and 63% for Chauffeur. While individual campaigns favor different architectures, these differences balance out in aggregate. This convergence suggests that extreme geometric difficulty produces prioritization signals that are less sensitive to architectural differences.

Frenetic_v Campaigns: In Frenetic_v scenarios yield the strongest prioritization performance, with mean $EFD-Top10$ of 88% for DAVE-2 and 89% for Chauffeur, and multiple campaigns achieving 100% detection within the first 10 tests. The slightly higher

mean for Chauffeur indicates that temporal behavioral features can better distinguish failure severity when scenarios are explicitly curated to induce faults.

APFD Analysis: While EFD captures early effectiveness, we further evaluate cumulative prioritization quality using the Average Percentage of Fault Detection (APFD). Across all campaigns, APFD values range from 0.82 to 0.97, with mean values of 0.92 for DAVE-2 and 0.91 for Chauffeur, indicating that faults are consistently concentrated early in the execution order.

Examining campaign types reveals complementary trends: Ambiegen campaigns show identical mean APFD (0.92) for both models, Frenetic campaigns slightly favor DAVE-2 (0.92 vs. 0.90), and Frenetic_v campaigns favor Chauffeur (0.93 vs. 0.90). These small differences ($\leq$0.03) reflect architecture-specific sensitivities rather than systematic bias, where certain geometric or dynamic characteristics differentially challenge feedforward versus recurrent control policies, but these campaign-level variations average out to produce equivalent overall prioritization effectiveness across the test suite.

Overall, the prioritization strategy, combining geometric complexity, dynamic behavioral difficulty, and historical failures consistently, concentrates faults early for both ADAS models, demonstrating robust generalization across architectures.

6.4.5.4 $\mathbf{RQ_{3d}}$ (Model Transferability)

Table 6.4 summarizes results across all 36 campaigns, enabling a systematic evaluation of how well test suites and failure patterns transfer between architecturally distinct ADAS models. The *Total* column indicates the aggregate number of simulation scenarios executed for each campaign. The *Selected Count* group reports the test selection, with the *Dave2* and *Chauffeur* columns displaying the number of test cases selected by each model. The *DAVE-2 Failures* column group tests failed by the DAVE-2; specifically, the *Total* column lists the absolute number of failures, while *by D2* and *by Ch* indicate whether those failures were selected by the DAVE-2 (self-selection) or the Chauffeur selection policy (cross-selection), respectively. Similarly, the *Chauffeur Failures* group details tests failed by the Chauffeur, listing the *Total* failures alongside those identified *by Ch* (self-selection) versus those identified *by D2* (cross-selection). Finally, the *Overlap* column reports the number of tests that are failed by both models (i.e., the intersection of their failed-test sets).

Selection Stability Across Models: We first examine whether the selection mechanism itself transfers across models. The framework shows high stability: on average, DAVE-2-based clustering selects 8.5% of roads, while Chauffeur-based clustering selects 8.8%, a difference of only 0.3 percentage points. Despite architectural differences, this consistency

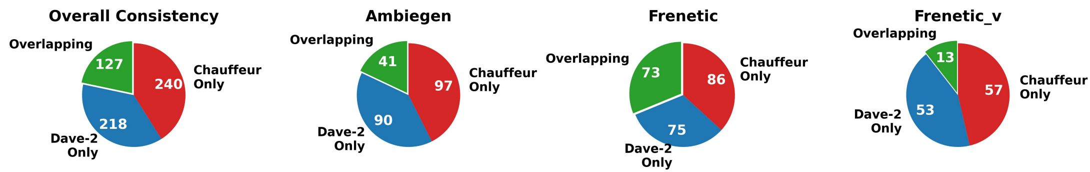


Figure 6.5: Cross-model test failure consistency analysis

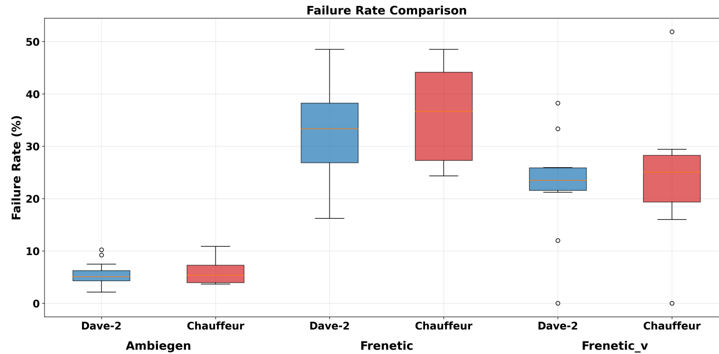


Figure 6.6: Failure retention across models (Dave-2/Chauffeur).

indicates that geometric features dominate clustering, while dynamic behavioral information refines selections rather than fundamentally reshaping them. However, as shown next, stable selection does not imply equivalent failure detection.

Failure Retention Across Models: As illustrated in Figure 6.6, across all campaigns, DAVE-2 exhibits 303 total failures (8.4 per campaign), while Chauffeur exhibits 333 failures (9.3 per campaign), representing a 9.9% increase for Chauffeur. Importantly, failure distributions vary by campaign type:

- **Ambiegen:** Chauffeur shows 125 failures versus 97 for DAVE-2 (+28.9%).
- **Frenetic:** Chauffeur shows 131 failures versus 145 for DAVE-2 (–9.7%).
- **Frenetic_v:** Chauffeur shows 77 failures versus 61 for DAVE-2 (+26.2%).

These patterns suggest that Chauffeur's recurrent architecture is more vulnerable in nominal and validated critical scenarios, while offering relative robustness under aggressive geometric stress, where temporal context aids control smoothing.

Own-Profile vs. Cross-Profile Effectiveness: We next evaluate whether a model's optimized test suite effectively exposes failures in the other model. Across all campaign types, selections based on a model's own behavioral profile consistently retain more failures than cross-model selections. For DAVE-2, own-profile selection retains 78.2% of failures, compared to 70.3% under Chauffeur-based selection (–7.9 pp). The degradation is most pronounced in Frenetic campaigns (-10.4 pp), indicating that aggressive geometric sequences amplify architecture-specific behavioral differences. For Chauffeur, own-profile

selection retains 78.4% of failures versus 74.2% under DAVE-2-based selection (–4.2 pp). While the effect is smaller, it is consistent across campaign types.

This asymmetry arises because behavioral difficulty metrics (e.g., steering variability, cross-track error, yaw rate variability) capture architecture-dependent responses rather than intrinsic road difficulty. A road challenging for frame-based control may be benign for a recurrent policy, and vice versa.

Failure Overlap and Model-Specific Vulnerabilities: To quantify shared versus model-specific failures, we compute Jaccard similarity coefficients Equation 6.17 between failure sets detected by each model's optimized selection. Across all campaigns, similarity ranges from 0.00 to 0.83, with a mean of 0.40, indicating that only 40% of detected failures are shared, while 60% are model-specific, as illustrated in Figure 6.5.

Campaign type strongly influences overlap. Frenetic campaigns exhibit the highest similarity (mean 0.52), suggesting that geometric extremes create architecture-agnostic challenges. Ambiegen and Frenetic_v campaigns show lower similarity (means 0.41 and 0.39), indicating greater sensitivity to architectural differences even in validated critical scenarios. Several campaigns exhibit complete disjunction (J = 0.00), where each model fails on entirely different roads despite identical geometric clustering. These cases demonstrate that behavioral profiling can expose fundamentally different vulnerability patterns depending on architecture. The complete disjunction (J = 0.00) may arise from fundamental differences in temporal information processing between feedforward and recurrent architectures. DAVE-2's feedforward CNN processes each frame independently, making it vulnerable to instantaneous perceptual ambiguities (e.g., poorly visible lane markings in a single frame, sudden lighting transitions) that produce immediate steering errors [101].

In contrast, Chauffeur's LSTM architecture maintains temporal context through its memory cells, enabling it to *smooth over* frame-level ambiguities by leveraging historical observations [299]. However, this temporal dependency introduces a distinct failure mode: Chauffeur can fail on roads where incorrect temporal patterns accumulate over sequences [300], such as gradually curving roads where early subtle misalignments compound into significant tracking errors. Conversely, DAVE-2 may successfully navigate these roads by responding reactively to each frame's immediate visual features without temporal bias. This architectural complementarity explains why scenarios challenging one model may be benign for the other, resulting in zero overlap despite both models being trained on similar data and tested under identical geometric conditions.

Bidirectional Transfer Asymmetry: To capture directional transferability, we compute conditional probabilities of failure overlap Equation 6.18. Across all campaigns, $P(F_C \mid F_D)$ averages 0.42, while $P(F_D \mid F_C)$ averages 0.38, indicating modest asymmetry

favoring DAVE-2 as a more predictive failure source. One plausible explanation is that DAVE-2's feedforward CNN architecture tends to expose fundamental perceptual challenges in individual frames, such as ambiguous visual cues or difficult lighting conditions that persist regardless of temporal modeling. These frame-level difficulties may therefore remain challenging even when Chauffeur applies temporal smoothing through its LSTM layers. In contrast, some of Chauffeur's failures may arise from temporal reasoning effects specific to recurrent architectures (e.g., sensitivity to accumulated prediction errors), which are not encountered by DAVE-2's stateless, frame-by-frame processing. As a result, failures observed in Chauffeur may be less indicative of scenarios that challenge fundamentally different model architectures.

This asymmetry is strongest in Frenetic campaigns (0.60 vs. 0.47), suggesting that DAVE-2's frame-based failures often correspond to roads that also stress Chauffeur, whereas Chauffeur exhibits additional temporal failure modes that DAVE-2 avoids. Ambiegen campaigns show moderate asymmetry, while Frenetic_v campaigns show near symmetry, indicating more balanced failure predictability in validated scenarios.

Practical Implications and Limitations: Overall, the results indicate moderate but incomplete cross-model transferability. High selection stability confirms geometric clustering as a robust, model-agnostic foundation, but only 70-74% of failures transfer across models, and Jaccard similarity shows that over half of detected failures are model-specific. When a single shared test suite is required, DAVE-2-based selection offers marginally better cross-model coverage, but the advantage is modest, and model-specific behavioral profiling remains essential for uncovering architecture-dependent failures.

These findings are subject to important limitations. The study evaluates only two vision-based imitation learning models within a deterministic simulator. Generalization to reinforcement learning, transformer-based policies, multi-modal sensing, or real-world environments with perceptual noise remains unvalidated. Moreover, OPENCAT-generated roads may underrepresent urban and low-speed scenarios where different architectural trade-offs could dominate.

Overall, the results across RQ_{3a}-RQ_{3d} demonstrate that integrating geometric complexity, dynamic behavioral variation, and historical failure information enables effective reduction, ordering, and partial transfer of ADAS test suites. The selection mechanism consistently preserves representative geometric and behavioral coverage while eliminating a large proportion of redundant roads ($RQ_{3a,3b}$), ensuring that safety-critical scenarios are retained despite substantial test suite reduction. Building on this foundation, the hybrid prioritization strategy reliably ranks high-risk roads early, yielding significant improvements in early fault detection when applied to the same model (RQ_{3c}). Although behavioral profiling is inherently model-specific, the selected and prioritized roads exhibit meaningful cross-model

effectiveness, revealing a substantial fraction of failures in alternative architectures while also exposing model-specific vulnerabilities (RQ_{3d}). Collectively, these findings indicate that the proposed framework supports cost-effective, failure-aware test suite construction that balances model-agnostic geometric structure with architecture-sensitive behavioral insights for autonomous driving evaluation.

Table 6.4: Failure detection comparison between DAVE-2 and Chauffeur models.

Campaign	**Total**	**Selected Count**		**DAVE-2 Failures**			**Chauffeur Failures**			**Overlap**
		Dave2	**Chauffeur**	**Total**	**by D2**	**by Ch**	**Total**	**by Ch**	**by D2**	
Ambeigen Campaigns										
2	973	147 (15%)	151 (16%)	11	5	5	16	7	7	9
3	964	206 (21%)	208 (22%)	9	8	9	11	10	9	3
4	965	178 (18%)	180 (19%)	5	5	4	7	6	6	1
5	958	167 (17%)	173 (18%)	10	8	7	10	8	7	3
6	959	179 (19%)	183 (19%)	9	7	6	13	8	7	2
7	963	197 (21%)	193 (20%)	10	7	8	9	4	5	3
8	952	176 (18%)	179 (19%)	11	10	11	14	10	9	3
9	953	183 (19%)	185 (19%)	4	4	4	7	7	7	0
10	952	176 (18%)	179 (19%)	11	10	11	14	10	9	3
11	953	183 (19%)	185 (19%)	4	4	4	7	7	7	0
13	965	178 (18%)	180 (19%)	5	5	4	7	6	6	1
14	958	167 (17%)	173 (18%)	10	8	7	10	8	7	3
15	959	179 (19%)	183 (20%)	9	7	6	13	8	7	2
Frenetic Campaigns										
2	928	27 (3%)	32 (4%)	7	4	4	7	4	4	3
3	954	41 (4%)	48 (5%)	11	8	8	13	8	7	7
4	964	29 (3%)	42 (4%)	12	7	9	13	11	8	7
5	945	30 (3%)	49 (5%)	8	5	5	14	10	6	4
6	944	33 (3%)	38 (4%)	16	7	7	16	7	7	0
7	967	38 (4%)	23 (2%)	14	7	7	15	5	9	7
8	952	30 (3%)	33 (4%)	10	6	3	11	6	6	6
9	964	37 (4%)	39 (4%)	9	6	6	10	8	8	5
11	866	37 (4%)	33 (3%)	11	7	9	11	9	8	5
12	956	39 (4%)	39 (4%)	17	12	10	16	9	10	10
13	959	34 (4%)	35 (4%)	13	7	10	15	10	9	8
14	866	33 (4%)	34 (4%)	11	8	8	9	5	6	5
15	870	37 (4%)	32 (3%)	12	8	5	9	5	6	6
Frenetic_v Campaigns										
2	944	31 (3%)	37 (4%)	7	6	5	8	8	7	1
4	525	25 (5%)	33 (6%)	3	2	3	4	3	4	1
5	940	21 (2%)	35 (4%)	7	7	5	6	6	6	2
6	764	22 (3%)	26 (4%)	5	5	4	6	5	3	2
7	47	8 (17%)	9 (19%)	0	0	0	0	0	0	0
11	953	33 (3%)	31 (3%)	8	7	6	8	6	6	1
12	942	27 (3%)	33 (4%)	7	5	5	14	13	7	2
13	951	34 (4%)	37 (5%)	7	6	6	10	9	7	2
14	934	36 (4%)	30 (3%)	9	8	6	8	8	6	1
15	949	33 (4%)	29 (3%)	7	6	6	6	4	6	1

6.5 Threats to Validity

This section discusses potential threats to the validity of our study and the extent to which they may affect the interpretation and generalizability of the results. Following established empirical software engineering guidelines, we consider threats related to internal validity (e.g., configuration and methodological choices), construct validity (e.g., feature representation and oracle design), external validity (e.g., task and simulator scope), and conclusion validity (e.g., evaluation metrics and failure distributions). Where possible, we describe mitigation strategies adopted in the experimental design and outline directions for future work to further reduce these threats.

Parameter Sensitivity and Configuration Choices: The proposed framework relies on a combination of geometric and dynamic behavioral features, along with associated thresholds and weighting parameters, to guide test selection and prioritization. While these parameters are informed by prior literature and default configurations, alternative choices could influence selection granularity or prioritization order. However, the consistent performance observed across multiple campaign types, architectures, and feature configurations (RQ_{3a}-RQ_{3c}) suggests that the core findings are not overly sensitive to specific parameter settings. Nonetheless, architecture-specific tuning of feature weights may further improve performance; at the same time, tuning weights per model introduces a risk of *overfitting* the prioritization to a specific ADAS/controller and benchmark, potentially inflating observed gains without improving generalization to new models or campaigns.

Behavioral Feedback and Circularity Effects: Our framework derives behavioral features by executing the ADAS model under test and subsequently uses these features to guide scenario selection and prioritization. This introduces a potential feedback loop, as the prioritization strategy is informed by the same model behavior it aims to evaluate. While this design choice reflects the goal of behavior-aware testing, it constitutes a threat to internal validity, as prioritization may emphasize scenarios that amplify known weaknesses of the reference model. We partially mitigate this threat through cross-model transfer experiments (RQ_{3d}), which show that behavior-derived prioritization retains effectiveness across different architectures, although not perfectly. Fully decoupling prioritization from the evaluated model, for example, through ensemble-based profiling or surrogate behavioral models, remains an important direction for future work.

Benchmark and Simulator Scope: The evaluation is constrained by the use of OpenCat-converted SensoDat scenarios and the Udacity simulator, which provide a controlled and scalable experimental environment but do not fully capture real-world driving complexity, including sensor noise, environmental variability, and interactive traffic dynam-

ics. Moreover, while RQ_{3d} demonstrates partial cross-model transferability, it also reveals that a substantial fraction of failures remain model-specific, limiting direct generalization across architectures. Despite these constraints, the ability to systematically execute and analyze tens of thousands of scenarios across diverse campaign types strengthens the relevance of the observed trends within the lane-keeping setting. Future work should extend validation to additional simulators, benchmarks, and higher-fidelity environments to reduce the gap to real-world deployment.

Task and Functional Scope Limitations: The study focuses exclusively on lane-keeping ADAS implemented via imitation learning, which limits the external validity of the findings. While lane-keeping is a foundational ADS function and a common benchmark task, other driving behaviors, such as lane changes, obstacle avoidance, or interaction with other agents, may exhibit different failure modes and behavioral dynamics. As a result, the effectiveness of behavior-aware selection and prioritization observed in this work may not directly transfer to more complex or decision-intensive driving tasks without adaptation. Extending the framework to additional ADS functionalities is necessary to assess its general applicability.

Oracle and Feature Fidelity: The validity of the results also depends on how well the selected geometric and dynamic features, along with similarity measures such as DTW, represent driving difficulty and failure-inducing conditions. While these features capture key aspects of control instability and behavioral diversity, they may not fully reflect perception-driven failures, rare edge cases, or semantic driving context (e.g., signage, lighting conditions, or interactions with other agents). To mitigate this limitation, we employ multiple evaluation metrics (reduction, fault retention, EFD, and APFD) and validate results across different architectures and campaign types. Nevertheless, incorporating richer behavioral descriptors, perception-level signals, and realistic noise models remains an important direction for improving construct fidelity.

Failure Definition and Severity Granularity: Failures are defined using a binary oracle based on out-of-bounds (OOB) events or invalid scenarios, consistent with prior work and the SensoDat benchmark. While this definition enables scalable and reproducible evaluation, it abstracts away differences in failure severity, recovery margins, and near-miss behaviors that may be equally relevant in safety-critical contexts. Consequently, scenarios that induce severe control instability without triggering OOB are treated as successful executions, potentially underestimating their testing value. This limitation affects construct validity and suggests that future work should explore graded failure notions or risk-aware oracles that better capture safety-relevant behavior.

Metric Sensitivity and Failure Distribution Bias: Finally, the evaluation relies on prioritization metrics such as EFD and APFD, which are sensitive to the distribution and sparsity of failures within a test suite. In campaigns with very low or very high failure rates, relative improvements in these metrics may be influenced by statistical effects rather than intrinsic prioritization quality. To mitigate this threat, we report results across multiple campaign types and focus on comparative trends rather than absolute metric values. Nonetheless, alternative evaluation protocols, such as stratified sampling or cost-aware effectiveness measures, could further strengthen conclusion validity.

This chapter introduced a behavior-aware test suite reduction and prioritization framework for ADAS regression testing. By integrating geometric properties of road segments with dynamic behavioral features extracted from execution traces, the framework addresses the scalability challenges posed by large-scale benchmark-based testing while preserving fault-detection capability. Extensive empirical evaluation across multiple driving environments and imitation-learning ADAS models demonstrates that the proposed approach substantially reduces execution cost and accelerates failure discovery compared to random, geometry-only, and behavior-only baselines. While the evaluation is subject to limitations related to task scope, simulator fidelity, and model diversity, the results provide strong evidence that incorporating behavioral feedback into test selection and prioritization enables more efficient and informative testing. These findings support the broader thesis argument that effective validation of DL-based ADAS requires moving beyond static scenario definitions toward behavior-centric testing methodologies, setting the stage for the synthesis and implications discussed in the following chapter.

Part III

Synthesis and Conclusion

Chapter 7

Conclusion and Future Work

The validation of ADS stands at a critical juncture. As these systems transition from research prototypes to deployed products affecting public safety, the adequacy of existing testing methodologies becomes not merely an academic concern but a matter of societal consequence. This thesis examined the state of testing practices for DL systems, investigated the challenges of sharing testing resources across development contexts, and developed methods to manage the scalability problems that emerge when such sharing succeeds. Taken together, these investigations reveal both encouraging progress and fundamental challenges that remain unresolved.

The research began with an empirical characterization of current testing practices in deep learning projects (RQ_1, Chapter 1), establishing that systematic validation of model behavior is often missing or incomplete and that code coverage practices are rarely applied to model-specific components. These findings motivated an investigation into whether safety-critical systems such as ADS, which rely heavily on pre-existing benchmarks and simulators, achieve more comprehensive validation. This investigation identified benchmark and simulator interoperability as a fundamental barrier to effective testing (RQ_2, Chapter 1), revealing that interoperability challenges extend far beyond format conversion. Finally, recognizing that successful interoperability enables access to massive collections of test scenarios, the research addressed the resulting scalability challenge by developing intelligent test prioritization methods that make large-scale testing computationally feasible while preserving fault detection capability (RQ_3, Chapter 1).

7.1 Key Contributions

7.1.1 Empirical Characterization of Deep Learning Testing Practices

At the empirical level, the systematic study of 300 Python deep learning projects presented in Chapter 4 establishes quantitative evidence about the state of testing practices in the open-source community [91]. The investigation revealed several critical findings:

- **Testing adoption**: While 77% of projects include test suites, only 55% implement model-specific tests despite having general testing infrastructure. Moreover, 23% of projects lack any systematic testing, operating without safeguards that software engineering practice has established over decades [91].
- **Limited non-functional testing**: Performance testing appears in only 9% of

projects, security testing in less than 1%, and smoke testing in 3%, despite their importance for production deployment.

- **Weak CI/CD integration**: Although 86.63% of projects employ automated testing frameworks, only 18% integrate automated test execution into continuous integration pipelines, limiting early defect detection, a critical gap for safety-critical systems such as autonomous driving [91].
- **Rapid test suite growth**: Analysis of representative projects shows test suites growing up to tenfold within ten releases, with substantial maintenance overhead, as 20-40% of test files change per release and 11-25% of commits relate to testing.

These findings extend prior work by focusing specifically on deep learning projects and examining how testing practices evolve over time rather than capturing static snapshots. Importantly, they establish why ADS represent a worst-case instance of the observed challenges. ADS integrates multiple learned components in safety-critical applications operating in open-world environments under regulatory scrutiny. The combination of architectural complexity, safety consequences, infinite input spaces, and legal requirements creates validation demands that exceed what individual development teams can satisfy independently. This motivates reliance on shared benchmarks and simulators, framing benchmark reuse not as a convenience but as a practical necessity.

7.1.2 OpenCat: Infrastructure for Cross-Platform Benchmark Reuse

At the infrastructure level, OpenCat, presented in Chapter 5, serves both as a practical mechanism for cross-platform benchmark reuse and as an experimental instrument for analyzing scenario transfer limitations [149]. Specifically, this subsection highlights three key findings:

- **Practical interoperability at scale:** OpenCat enables the conversion of large-scale industrial benchmarks across simulators.
- **Geometric fidelity does not imply behavioral equivalence:** Accurate road geometry alone is insufficient to preserve test outcomes.
- **Benchmark semantics are execution-context dependent:** Scenario effectiveness depends on simulator dynamics and model architecture.

OpenCat converts 32,580 SensoDat scenarios from OpenDRIVE into Catmull-Rom spline representations with near-perfect geometric fidelity, removing a major barrier that previously prevented researchers using Udacity and similar academic simulators from accessing this benchmark. This establishes, for the first time, practical large-scale reuse of industry-grade road scenarios across heterogeneous simulation platforms.

However, the empirical evaluation reveals a critical limitation: *geometric fidelity alone does not guarantee behavioral fidelity.* Despite accurate geometric conversion, we observe a 25-50% variance in pass/fail outcomes between original and converted scenarios. This result challenges a common implicit assumption in ADS benchmarking, that geometric specifications sufficiently encode testing intent. The observed divergence indicates that test scenarios implicitly depend on execution context, including simulator dynamics, vehicle models, and control-loop characteristics. As a result, syntactic interoperability does not ensure semantic equivalence of tests across platforms.

Overall, this work contributes both a reusable infrastructure for benchmark sharing and empirical evidence that architecture-agnostic benchmarking requires more than format-level compatibility. These findings motivate the need for behavior-aware specifications that capture expected functional challenges rather than relying solely on geometric layouts.

7.1.3 Coverage-Guided Prioritization Framework

At the methodological level, the coverage-guided prioritization framework developed in Chapter 6 demonstrates that intelligent test selection can substantially improve testing efficiency without proportionally sacrificing fault detection capability [134]. The framework achieves:

- **Efficient reduction**: An average test suite reduction of 89% (up to 97%) while retaining 79% of failure-revealing scenarios.
- **Accelerated fault detection**: Failures are detected approximately 95 times faster than with random ordering.
- **Effective prioritization**: Multi-criteria prioritization based on geometric complexity, dynamic driving behavior, and historical failures yields APFD scores between 0.82 and 0.97.
- **Cross-model transferability**: Scenarios challenging one lane-keeping model (DAVE-2) frequently challenge another (Chauffeur), despite architectural differences.

Beyond these quantitative results, the framework demonstrates how geometric and behavioral features can be combined to characterize scenario difficulty. By focusing on what scenarios demand of systems rather than which code paths they execute, the approach shifts coverage from a structural to a behavioral perspective. The partial cross-model transferability observed suggests that some aspects of scenario difficulty reflect fundamental perception and control challenges rather than model-specific artifacts.

7.2 Implications for Research and Practice

The findings of this thesis have implications that extend beyond the specific tools and techniques developed, informing how testing for autonomous systems should be conceptualized and interpreted.

7.2.1 Model-Specific Testing as a Necessity

The empirical findings demonstrate that inherited software testing practices are insufficient for safety-critical learning-enabled systems. Testing infrastructure alone does not ensure meaningful validation when learned components remain largely untested. This implies that model-specific testing must be treated as a first-class concern in ADS development rather than an optional extension of software testing practices.

7.2.2 Behavioral Fidelity as a Coverage Dimension

A central implication of this work is the distinction between geometric fidelity and behavioral fidelity in simulation-based testing. The observed divergence between geometrically identical scenarios executed in different contexts demonstrates that structural diversity does not guarantee functional diversity. As a result, coverage metrics that operate purely over scenario geometry risk overestimating validation completeness. Effective testing must therefore consider coverage over behavioral space, including perception ambiguity, control difficulty, and planning complexity.

7.2.3 Context Coupling and Benchmark Interpretation

The demonstrated sensitivity of test outcomes to execution context implies that benchmark results should be interpreted with greater caution. Performance reported on a benchmark reflects not only system capability but also the assumptions embedded in the simulator, vehicle model, and execution environment. This challenges implicit claims of generalizability and suggests that benchmark results should be understood as context-dependent evidence rather than absolute performance indicators.

7.2.4 Scalability Requires Selection, Not Exhaustion

Finally, the success of coverage-guided prioritization implies that exhaustive execution is neither feasible nor necessary for effective validation at scale. Intelligent selection based on behavioral relevance enables practical regression testing even as test suites grow. This insight generalizes beyond autonomous driving to other domains such as robotics, aerospace systems, or industrial process control, where systems are validated through simulation rather than direct code execution.

7.3 Limitations of the Presented Work

While this thesis makes substantial contributions, several limitations warrant acknowledgment, as they constrain the scope of the findings and inform directions for future investigation.

7.3.1 Scope and Generalizability Constraints

The empirical study presented in Chapter 4 examines 300 Python-based deep learning projects from GitHub selected using popularity metrics [91]. While this design enables large-scale analysis, it introduces several limitations that constrain generalizability:

- **Project selection bias**: The focus on popular open-source repositories favors projects with established communities and higher visibility. Such projects may differ substantially from proprietary industrial codebases, where commercial pressures, liability considerations, and regulatory requirements can lead to more rigorous or differently structured testing practices. As a result, the observed testing maturity may not reflect practices in industrial or safety-certified environments.
- **Language and ecosystem restriction**: The exclusive focus on Python-based projects excludes DL systems implemented in languages such as C, C++, or Java, which may exhibit distinct testing cultures, tooling ecosystems, and quality assurance practices, particularly in embedded or real-time settings.
- **Observability of testing effectiveness**: The methodology relies on static repository analysis and version history examination to infer testing practices. While this enables scalable quantitative assessment, it cannot capture qualitative aspects of test effectiveness. The presence of tests, automation, or coverage indicators does not guarantee meaningful validation of model behavior. Conversely, projects with limited observable testing artifacts may rely on alternative quality assurance practices, such as extensive manual validation or internal verification processes, which remain invisible to repository-based analysis.

Overall, the findings characterize observable testing artifacts rather than the true effectiveness of quality assurance activities, and should be interpreted as indicators of testing maturity rather than definitive measures of software quality.

7.3.2 Interoperability and Transfer Limitations

The interoperability investigation in Chapter 5 focuses on converting OpenDRIVE road descriptions to a Catmull–Rom spline representation [149]. While this enables systematic cross-platform evaluation, several limitations affect the interpretation of the results:

- **Limited format coverage**: The OpenDRIVE-to-Catmull–Rom conversion represents only one instance of a broader and heterogeneous interoperability landscape. Other scenario representations (e.g., Lanelet2, ASAM OpenSCENARIO, proprietary formats) may introduce different semantic gaps, constraints, or failure modes that are not captured in this study.
- **Entangled sources of behavioral divergence**: The experimental comparison between original SensoDat scenarios executed in BeamNG.tech and converted scenarios executed in Udacity changes multiple aspects of the execution context simultaneously. In addition to the road representation, the simulator physics engine, vehicle dynamics model, sensor implementation, and autonomous driving controller all differ.
- **Model and control paradigm mismatch**: The learning-based DAVE-2 model used in the converted scenarios fundamentally differs from BeamNG's built-in AI driver, which relies on PID control with access to global road knowledge. Consequently, observed behavioral differences cannot be attributed solely to representation conversion, but rather reflect compounded effects of architectural and control-strategy differences.

These factors make it difficult to isolate individual causes of behavioral divergence. However, this entanglement also reflects realistic reuse scenarios, where benchmarks are rarely transferred in isolation from changes in the simulator and system architecture.

7.3.3 Prioritization Framework Constraints

The prioritization framework developed in Chapter 6 operates on OpenCat-converted SensoDat scenarios and is evaluated using two lane-keeping models, DAVE-2 and Chauffeur, both based on imitation learning [134]. This evaluation context introduces several limitations affecting the generalizability of the results:

- **Lane-keeping focus**: The framework targets Level 2 ADAS lane-keeping as a controlled, foundational ADS function, which represents only a narrow subset of autonomous driving capabilities. Higher levels of autonomy (SAE Levels 3-5) involve urban navigation, multi-agent interaction, and complex decision-making processes that are not addressed [44]. Perception tasks such as object detection, traffic sign recognition, and semantic segmentation may respond differently to scenario characteristics, potentially requiring alternative prioritization features. Similarly, planning and decision-making components for intersections, lane changes, and traffic interaction introduce challenges not captured by lane-keeping validation. The behavioral features used in this work, including steering variability, cross-track error, speed variation, and yaw rate changes, characterize lane-keeping difficulty but may not extend naturally to these other functions.

- **Architectural similarity**: Cross-model validation using DAVE-2 and Chauffeur provides evidence of partial generalization but remains limited by architectural similarity. Both models implement convolutional, imitation-learning-based control trained on human demonstrations. Autonomous systems based on planning-centric, rule-based, or hybrid architectures may exhibit different failure patterns, potentially reducing the predictive power of the selected behavioral features.

- **Semantic success criteria**: The framework uses out-of-bounds (OOB) detection as the primary test oracle. While widely adopted, OOB does not capture all safety-relevant failures, such as excessive steering oscillations, discomfort-inducing maneuvers, or near-miss events that remain within lane boundaries [134, 32].

- **Cold-start problem**: Behavioral prioritization requires initial execution of scenarios to extract behavioral features, limiting applicability for newly developed autonomous systems without execution history. While this limitation is acceptable for regression testing, where systems undergo frequent updates, it restricts use for initial validation of entirely new systems. In such cases, teams must either execute the full test suite initially or rely on geometric features alone until behavioral data becomes available. Alternative approaches like transfer learning from similar systems or lightweight simulation [301], enabling rapid behavioral profiling, might mitigate this limitation but require investigation beyond the current work's scope.

- **Parameter sensitivity**: The framework involves several tuning decisions, including curvature thresholds ($\kappa = 0.015$ m^{-1}), dynamic-geometric weighting ($w_{\text{dyn}} = 0.5$), and clustering linkage criteria. Although informed by standards and empirical evaluation [235], systematic ablation studies and practitioner-oriented guidance on parameter selection remain limited.

7.3.4 Simulation-to-Reality Gap

As with all simulation-based validation efforts, this work does not comprehensively address the simulation-to-reality gap. All evaluations are conducted in virtual environments, and the study does not validate whether failures observed in simulation correspond to failures that would occur in physical vehicles. Prior research has documented cases where models that perform well in simulation fail when deployed on real hardware due to unmodeled dynamics, sensor characteristics, or environmental factors [302, 303, 31]. While simulation enables systematic exploration of diverse and hazardous scenarios that are infeasible to test physically, ultimate validation of ADS requires complementary track testing and real-world deployment.

7.4 Future Research Directions

The limitations identified in this thesis motivate several directions for future investigation.

7.4.1 Extending Empirical Studies of Testing Practice

Future work should extend empirical analysis beyond open-source Python projects to proprietary industrial code bases and other programming languages. Partnerships with automotive manufacturers, autonomous driving technology companies, or research laboratories could enable analysis of commercial deep learning projects subject to regulatory requirements and liability concerns. Such analysis would reveal whether open-source findings generalize to industrial contexts or whether commercial development exhibits fundamentally different testing cultures. Longitudinal and qualitative studies could complement repository analysis by capturing decision-making processes and testing the rationale that is not visible in code artifacts.

7.4.2 Advancing Behavioral Benchmarking

Addressing the limitations of geometric benchmarking requires the development of behavioral scenario specification languages that encode expected functional challenges independently of simulator implementation. Systematic cross-simulator and cross-architecture studies are needed to identify which behavioral properties transfer reliably and which remain context-dependent.

7.4.3 Improving Test Selection and Cold-Start Prioritization

Future research should address the cold-start limitation of behavior-aware prioritization. Transfer learning, zero-shot prioritization using geometric proxies, and lightweight simulation techniques represent promising directions. Extending prioritization to multi-agent and planning-intensive scenarios will require new behavioral features capturing interaction and decision complexity beyond steering variability, cross-track error, speed variation, and yaw rate changes. Additionally, adaptive prioritization that updates rankings based on accumulated execution results could improve efficiency for iterative testing workflows. Rather than computing priority once and executing tests in fixed order, adaptive approaches might re-rank remaining tests after each execution based on observed outcomes. If early tests reveal unexpected failure patterns, subsequent prioritization might emphasize similar scenarios to explore failure boundaries. Reinforcement learning formulations where prioritization agents receive feedback from execution outcomes and learn to select maximally informative tests represent a promising direction connecting test prioritization to active learning and experiment design.

7.4.4 Closing the Simulation-to-Reality Gap

Systematic sim-to-real validation studies are required to determine which simulated failures correspond to real-world risks. Integrating real-world incident data into test selection and prioritization frameworks could further align testing with operational safety concerns.

7.4.5 Extending to Complex ADAS Functions and Additional Simulator Formats

The behavior-aware test prioritization framework can be extended beyond lane-keeping to more complex ADAS functions. While the current approach relies on features such as steering variability, cross-track error, and yaw dynamics, other functions, such as Automated Emergency Braking and Adaptive Cruise Control, require different behavioral descriptors, including deceleration profiles, time-to-collision estimates, object detection confidence, and multi-object interaction metrics. Developing a systematic methodology for defining function-specific behavioral features would enable the framework to generalize across ADAS functionalities, while preserving its core prioritization mechanisms, which remain largely function-agnostic.

Additionally, the interoperability infrastructure can be extended by supporting additional conversions between scenario formats. Extending the current OpenDRIVE-based pipeline to formats such as OpenSCENARIO would enable execution across widely used simulators (e.g., CARLA), thereby increasing the portability and adoption of the benchmark scenarios.

7.5 Closing Remarks

ADS represent one of the most safety-critical applications of deep learning and artificial intelligence. As these systems progress from research prototypes to production deployment, the demands on validation and testing become increasingly stringent. This thesis examined autonomous driving system validation through empirical analysis, infrastructure development, and methodological innovation. The results show that current testing practices fall short of safety-critical requirements, that benchmark reuse introduces subtle but consequential challenges, and that scalability demands behavior-aware test selection. These findings provide both practical contributions that teams can adopt immediately and conceptual insights informing how we think about validation for learned systems. Addressing these challenges requires not only improved tools but also a shift in how validation evidence is interpreted. By clarifying these issues and proposing practical paths forward, this work contributes toward more effective and trustworthy validation of ADS.

Bibliography

[1] A. Acharya. (2017, jan) How self-driving cars work - architecture overview. Accessed: 2026-01-19. [Online]. Available: https://atul.fyi/post/2017/01/03/how-self-driving-cars-work/

[2] SAE International, "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles," SAE International, Standard J3016_202104, April 2021, sAE J3016. [Online]. Available: https://www.sae.org/standards/content/j3016/

[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[4] I. D. Mienye and T. G. Swart, "A comprehensive review of deep learning: Architectures, recent advances, and applications," *Information*, vol. 15, no. 12, p. 755, 2024.

[5] M. H. M. Noor and A. O. Ige, "A survey on state-of-the-art deep learning applications and challenges," *Engineering Applications of Artificial Intelligence*, vol. 159, p. 111225, 2025.

[6] R. Valentin, "Towards a framework for deep learning certification in safety-critical applications using inherently safe design and run-time error detection," *arXiv preprint arXiv:2403.14678*, 2024.

[7] H. Forsberg, J. Lindén, J. Hjorth, T. Månefjord, and M. Daneshtalab, "Challenges in using neural networks in safety-critical applications," in *Proceedings of the 2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*. IEEE, 2020, pp. 1–7.

[8] A. E. Goodloe, " Assuring Safety-Critical Machine Learning-Enabled Systems: Challenges and Promise ," *Computer*, vol. 56, no. 09, pp. 83–88, 2023. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/MC.2023.3266860

[9] A. Esteva, A. Robicquet, B. Ramsundar, V. Kuleshov, M. DePristo, K. Chou, C. Cui, G. Corrado, S. Thrun, and J. Dean, "A guide to deep learning in healthcare," *Nature Medicine*, vol. 25, no. 1, pp. 24–29, 2019. [Online]. Available: https://www.nature.com/articles/s41591-018-0316-z

[10] X. Huang, D. Kroening, W. Ruan, J. Sharp, Y. Sun, E. Thamo, M. Wu, and X. Yi, "A survey of safety and trustworthiness of deep neural networks: Verification, testing,

adversarial attack and defence, and interpretability," *Computer Science Review*, vol. 37, p. 100270, 2020.

[11] N. Humbatova, G. Jahangirova, G. Bavota, V. Riccio, A. Stocco, and P. Tonella, "Taxonomy of real faults in deep learning systems," in *Proceedings of the ACM/IEEE 42nd international conference on software engineering*, 2020, pp. 1110–1121.

[12] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Transactions on Software Engineering*, vol. 48, no. 1, pp. 1–36, 2022.

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

[14] P. Koopman and M. Wagner, "Autonomous vehicle safety: An interdisciplinary challenge," *IEEE Intelligent Transportation Systems Magazine*, vol. 9, no. 1, pp. 90–96, 2017.

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

[16] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, 2020.

[17] 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.

[18] National Highway Traffic Safety Administration, "Standing general order on crash reporting for level 2 advanced driver assistance systems," U.S. Department of Transportation, Tech. Rep., 2022. [Online]. Available: https://www.nhtsa.gov/sites/nhtsa.gov/files/2022-06/ADAS-L2-SGO-Report-June-2022.pdf

[19] 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.

[20] N. Kalra and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?" *Transportation Research Part A: Policy and Practice*, vol. 94, pp. 182–193, 2016.

[21] Connected Automated Driving, "Recommendations for a european framework for testing on public roads: Regulatory roadmap for automated driving," Web article / Blog, 2023, policy / recommendations web page. [Online]. Available: https://www.connectedautomateddriving.eu/blog/recommendations-for-a-european-framework-for-testing-on-public-roads-regulatory-roadmap-for-au

[22] OPAL-RT Technologies Inc., "5 proven ways to validate autonomous driving algorithms with simulation," Blog post, 2024, vendor/industry blog post — web resource. [Online]. Available: https://www.opal-rt.com/blog/5-proven-ways-to-validate-autonomous-driving-algorithms-with-simulation/

[23] P. Koopman and M. Wagner, "Challenges in autonomous vehicle testing and validation," *SAE International Journal of Transportation Safety*, vol. 4, no. 1, pp. 2016–01–0128, apr 2016. [Online]. Available: https://www.sae.org/articles/challenges-autonomous-vehicle-testing-validation-2016-01-0128

[24] National Highway Traffic Safety Administration, "A framework for automated driving system testable cases and scenarios," U.S. Department of Transportation, Tech. Rep., 2018. [Online]. Available: https://rosap.ntl.bts.gov/view/dot/38824

[25] ——, "Standing general order on crash reporting: Initial data release," U.S. Department of Transportation, Tech. Rep., 2022.

[26] T. Bein, H. Atzrodt, R. Bartolozzi, S. Kupjetz, J. Millitzer, J. Nuffer, M. Rauschenbach, and G. Stoll, "Verification and validation of automated driving systems utilizing probabilistic fmea and simulation approaches," *Transportation Research Procedia*, vol. 72, pp. 470–477, 2023, tRA Lisbon 2022 Conference Proceedings Transport Research Arena (TRA Lisbon 2022),14th-17th November 2022, Lisboa, Portugal. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352146523007263

[27] V. Riccio, G. Jahangirova, A. Stocco, N. Humbatova, M. Weiss, and P. Tonella, "Testing machine learning based systems: a systematic mapping," *Empirical Software Engineering*, vol. 25, no. 6, pp. 5193–5254, 2020.

[28] A. Stocco, B. Pulfer, and P. Tonella, "Mind the Gap! A Study on the Transferability of Virtual Versus Physical-World Testing of Autonomous Driving Systems," *IEEE Transactions on Software Engineering*, vol. 49, no. 04, pp. 1928–1940, apr 2023.

[29] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Conference on robot learning*. PMLR, 2017, pp. 1–16.

[30] BeamNG.tech, "BeamNG GmbH," https://beamng.tech/, 2024, online; accessed 2024-11-19.

[31] M. Borg, R. B. Abdessalem, S. Nejati, F.-X. Jegeden, and D. Shin, "Digital twins are not monozygotic–cross-replicating adas testing in two industry-grade automotive simulators," in *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2021, pp. 383–393.

[32] C. Birchler, S. Khatiri, P. Derakhshanfar, S. Panichella, and A. Panichella, "Single and multi-objective test cases prioritization for self-driving cars in virtual environments," *ACM Transactions Software Engineering and Methodology.*, vol. 32, no. 2, Apr. 2023. [Online]. Available: https://doi.org/10.1145/3533818

[33] I. H. Sarker, "Deep learning: a comprehensive overview on techniques, taxonomy, applications and research directions," *SN Computer Science*, vol. 2, no. 6, p. 420, 2021.

[34] W. M. A. dos Santos, "Understanding the testing culture of machine learning projects on github." 2023. [Online]. Available: http://dspace.sti.ufcg.edu.br:8080/xmlui/handle/riufcg/29359

[35] R. M. da Silva, C. Cruz, H. de S. Campos, L. G. Murta, and V. de Oliveira Neves, "What is the adoption level of automated support for testing in open-source ecosystems?" in *Proceedings of the IV Brazilian Symposium on Systematic and Automated Software Testing*, 2019, pp. 80–89.

[36] A. Islam, N. T. Hewage, A. A. Bangash, and A. Hindle, "Evolution of the practice of software testing in java projects," in *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*. IEEE, 2023, pp. 367–371.

[37] P. Zhang, Y. Wang, X. Liu, Z. Lu, Y. Yang, Y. Li, L. Chen, Z. Wang, C.-A. Sun, X. Yu, and Y. Zhou, "Assessing effectiveness of test suites: What do we know and what should we do?" *ACM Transactions Software Engineering and Methodology.*, vol. 33, no. 4, Apr. 2024. [Online]. Available: https://doi.org/10.1145/3635713

[38] J. Li, R. Xu, X. Liu, J. Ma, B. Li, Q. Zou, J. Ma, and H. Yu, "Domain adaptation based object detection for autonomous driving in foggy and rainy weather," *IEEE Transactions on Intelligent Vehicles*, 2024.

[39] D. E. Rzig, A. Houerbi, R. G. Chavan, and F. Hassan, "Empirical analysis on ci/cd pipeline evolution in machine learning projects," *arXiv preprint arXiv:2403.12199*, 2024.

[40] C. Birchler, C. Rohrbach, T. Kehrer, and S. Panichella, "Sensodat: Simulation-based sensor dataset of self-driving cars," in *Proceedings of the 21st International Conference on Mining Software Repositories*, ser. MSR '24. New York, NY,

USA: Association for Computing Machinery, 2024, p. 510–514. [Online]. Available: https://doi.org/10.1145/3643991.3644891

[41] C. Lu, T. Yue, and S. Ali, "Deepscenario: An open driving scenario dataset for autonomous driving system testing," in *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, 2023, pp. 52–56.

[42] J. Dai, B. Gao, M. Luo, Z. Huang, Z. Li, Y. Zhang, and M. Yang, "Sctrans: Constructing a large public scenario dataset for simulation testing of autonomous driving systems," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3597503.3623350

[43] V. Nenchev, "One stack, diverse vehicles: Checking safe portability of automated driving software," in *2025 IEEE/SICE International Symposium on System Integration (SII)*. IEEE, 2025, pp. 764–769.

[44] S. Tang, Z. Zhang, Y. Zhang, J. Zhou, Y. Guo, S. Liu, S. Guo, Y.-F. Li, L. Ma, Y. Xue *et al.*, "A survey on automated driving system testing: Landscapes and trends," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 5, pp. 1–62, 2023.

[45] Y. Liao, J. Zhang, J. Keung, Y. Xiao, and Y. Dai, "Advancing autonomous driving system testing: Demands, challenges, and future directions," *Information and Software Technology*, p. 107859, 2025.

[46] M. Dupuis, M. Strobl, and H. Grezlikowski, "OpenDRIVE 2010 and beyond – status and future of the de facto standard for the description of road networks," in *Proceeding of the Driving Simulation Conference Europe*, 2010, pp. 231–242.

[47] Udacity, "Self-driving car simulator," 2023, accessed: 2025-11-04. [Online]. Available: https://github.com/udacity/self-driving-car-sim

[48] M. K. Ahuja, A. Gotlieb, and H. Spieker, "Testing deep learning models: A first comparative study of multiple testing techniques," in *2022 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2022, pp. 130–137.

[49] Y. Deng, X. Zheng, M. Zhang, G. Lou, and T. Zhang, "Scenario-based test reduction and prioritization for multi-module autonomous driving systems," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 82–93.

[50] M. Müller, *Dynamic Time Warping.* Springer Berlin Heidelberg, 2007, pp. 69–84. [Online]. Available: https://doi.org/10.1007/978-3-540-74048-3_4

[51] J. H. Ward Jr, "Hierarchical grouping to optimize an objective function," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 236–244, 1963.

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

[53] Team Chauffeur, "Steering angle model: Chauffeur," https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models/chauffeur, 2016, accessed: [14-10-2025]. [Online]. Available: https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models/chauffeur

[54] L. Baresi and M. Pezze, "An introduction to software testing," *Electronic Notes in Theoretical Computer Science*, vol. 148, no. 1, pp. 89–111, 2006.

[55] Larion, "Software testing 101: Types, methodologies, models & stlc," https://larion.com/software-testing-fundamentals/, 2024, blog post; Accessed: 2025-11-08.

[56] K. Sneha and G. M. Malle, "Research on software testing techniques and software automation testing tools," in *2017 international conference on energy, communication, data analytics and soft computing (ICECDS).* IEEE, 2017, pp. 77–81.

[57] G. J. Myers, C. Sandler, and T. Badgett, *The art of software testing*, 3rd ed. John Wiley & Sons, Ltd, 2011, ch. 5,6,7.

[58] Testlio, "Ci/cd and the need for test automation," 2024, blog post; Accessed: 2025-11-08. [Online]. Available: https://testlio.com/blog/ci-cd-test-automation/

[59] H. Wang, S. Yu, C. Chen, B. Turhan, and X. Zhu, "Beyond accuracy: an empirical study on unit testing in open-source deep learning projects," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 4, pp. 1–22, 2024.

[60] 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.

[61] M. Issler, Q. Goss, and M. İ. Akbaş, "Complexity evaluation of test scenarios for autonomous vehicle safety validation using information theory," *Information*, vol. 15, no. 12, p. 772, 2024.

[62] S. Segura, G. Fraser, A. B. Sanchez, and A. Ruiz-Cortés, "A survey on metamorphic testing," *IEEE Transactions on Software Engineering*, vol. 42, no. 9, pp. 805–824, 2016.

[63] L. Ma *et al.*, "Metamorphic testing for machine learning," *IEEE Transactions on Software Engineering*, 2019.

[64] A. E. Goodloe, " Assuring Safety-Critical Machine Learning-Enabled Systems: Challenges and Promise ," *Computer*, vol. 56, no. 09, pp. 83–88, Sep. 2023. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/MC.2023.3266860

[65] B. Potter and G. McGraw, "Software security testing," *IEEE Security & Privacy*, vol. 2, no. 5, pp. 81–85, 2004.

[66] Z. Xu, X. Wang, X. Wang, and N. Zheng, "Safety validation for connected autonomous vehicles using large-scale testing tracks in high-fidelity simulation environment," *Accident Analysis & Prevention*, vol. 215, p. 108011, 2025.

[67] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2015.

[68] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *IEEE Symposium on Security and Privacy*, 2017.

[69] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *Artificial Intelligence Safety and Security*, 2017.

[70] The MathWorks, Inc. (2026) What is smoke testing? The MathWorks, Inc. Accessed: 2026-01-27. [Online]. Available: https://www.mathworks.com/discovery/smoke-testing.html

[71] C. Cannavacciuolo and L. Mariani, "Smoke testing of cloud systems," in *Proceedings of the International Conference on Software Testing, Verification and Validation*, 2022, pp. 47–57.

[72] M. Fowler and M. Foemmel, "Continuous integration," *Thought-Works*, 2006.

[73] 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.

[74] J. Humble and D. Farley, *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

[75] D. Jeffrey and N. Gupta, "Improving fault detection capability by selectively retaining test cases during test suite reduction," *IEEE Transactions on software Engineering*, vol. 33, no. 2, pp. 108–123, 2007.

[76] M. J. Harrold, R. Gupta, and M. L. Soffa, "A methodology for controlling the size of a test suite," vol. 2, no. 3. ACM, 1993, pp. 270–285.

[77] A. Vahabzadeh, A. Stocco, and A. Mesbah, "Fine-grained test minimization," in *Proceedings of the 40th ACM/IEEE International Conference on Software Engineering*, ser. ICSE 2018. ACM, may 2018, pp. 210–221.

[78] G. Rothermel, M. J. Harrold, J. Von Ronne, and C. Hong, "Empirical studies of test-suite reduction," *Software Testing, Verification and Reliability*, vol. 12, no. 4, pp. 219–249, 2002.

[79] G. Rothermel, R. Untch, C. Chu, and M. Harrold, "Prioritizing test cases for regression testing," *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 929–948, 2001.

[80] S. Elbaum, G. Rothermel, and J. Penix, "Techniques for improving regression testing in continuous integration development environments," pp. 235–245, 2014.

[81] 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.

[82] 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.

[83] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test case prioritization: A family of empirical studies," *IEEE Transactions on Software Engineering*, vol. 28, no. 2, pp. 159–182, 2002.

[84] H. Forsberg, J. Lindén, J. Hjorth, T. Månefjord, and M. Daneshtalab, "Challenges in using neural networks in safety-critical applications," in *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*. IEEE, 2020, pp. 1–7.

[85] L. Tamang, M. R. Bouadjenek, R. Dazeley, and S. Aryal, "Handling out-of-distribution data: A survey," *IEEE Transactions on Knowledge and Data Engineering*, 2025.

[86] A.-R. Nuhu *et al.*, "A validation strategy for deep learning models: Evaluating and enhancing robustness," *IEEE Transactions on Neural Networks and Learning Systems*, 2025.

[87] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, pp. 1–48, 2019.

[88] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," 2014. [Online]. Available: https://arxiv.org/abs/1312.6199

[89] J. Sekhon and C. Fleming, "Towards improved testing for deep learning," in *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER).* IEEE, 2019, pp. 85–88.

[90] K. Shen *et al.*, "Munn: Mutation analysis of neural networks," in *ASE Workshops*, 2018.

[91] Q. Ali, O. Riganelli, and L. Mariani, "Testing in the evolving world of dl systems: Insights from python github projects," in *2024 IEEE 24th International Conference on Software Quality, Reliability and Security (QRS).* IEEE, 2024, pp. 25–35.

[92] T. Menzel, G. Bagschik, and M. Maurer, "Scenarios for development, test and validation of automated vehicles," in *2018 IEEE intelligent vehicles symposium (IV).* IEEE, 2018, pp. 1821–1827.

[93] *Road vehicles – Functional safety*, International Standard ISO 26262, International Organization for Standardization Standard, 2018, version: 2018-12. [Online]. Available: https://www.iso.org/standard/77490.html

[94] International Organization for Standardization, *Road vehicles – Safety of the intended functionality*, International Organization for Standardization Standard ISO 21 448:2022, Jun. 2022, second edition. [Online]. Available: https://www.iso.org/standard/77490.html

[95] *ISO/SAE 21434: Road Vehicles — Cybersecurity Engineering*, International Organization for Standardization and SAE International Std. 21 434, 2021.

[96] UNECE Working Party on Automated/Autonomous and Connected Vehicles (GRVA), "Global Technical Regulation on the Safety of Automated Driving Systems," United Nations Economic Commission for Europe (UNECE), Tech. Rep., 2025, uN GTR No. XX on ADS Safety.

[97] X. Wang, M. A. Maleki, M. W. Azhar, and P. Trancoso, "Moving forward: A review of autonomous driving software and hardware systems," *arXiv preprint arXiv:2411.10291*, 2024.

[98] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[99] Y. Zhang, A. Carballo, H. Yang, and K. Takeda, "Perception and sensing for autonomous vehicles under adverse weather conditions: A survey," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 196, pp. 146–177, 2023.

[100] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, vol. 25, 2012.

[101] Z. Chen and X. Huang, "End-to-end learning for lane keeping of self-driving cars," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 1856–1860.

[102] L. Chen, P. Wu, K. Chitta, B. Jaeger, A. Geiger, and H. Li, "End-to-end autonomous driving: Challenges and frontiers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[103] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2722–2730.

[104] 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.

[105] T. Bein, H. Atzrodt, R. Bartolozzi, S. Kupjetz, J. Millitzer, J. Nuffer, M. Rauschenbach, and G. Stoll, "Verification and validation of automated driving systems utilizing probabilistic fmea and simulation approaches," *Transportation research procedia*, vol. 72, pp. 470–477, 2023.

[106] S. S. Shetiya, V. Vyas, and S. Renukuntla, "Verification and validation of autonomous systems," *arXiv preprint arXiv:2411.13614*, 2024.

[107] C. Li, J. Sifakis, R. Yan, and J. Zhang, "Testing autonomous driving systems–what really matters and what doesn't," *arXiv preprint arXiv:2507.13661*, 2025.

[108] G. Rong, B. H. Shin, H. Tabatabaee, Q. Lu, S. Lemke, M. Možeiko, E. Boise, G. Uhm, M. Gerow, S. Mehta *et al.*, "Lgsvl simulator: A high fidelity simulator for autonomous driving," in *2020 IEEE 23rd International conference on intelligent transportation systems (ITSC)*. IEEE, 2020, pp. 1–6.

[109] T. Kramer, E. Murphy, C. Anderson, R. Ravikumar, A. Conway *et al.*, "Donkeycar: an open source hardware and software platform to build a small scale self driving car," https://www.donkeycar.com/, 2023, accessed: 2023-11-23.

[110] F. Yang, Y. Lu, B. Chen, P. Qin, and X. Peng, "Roadgen: Generating road scenarios for autonomous vehicle testing," *arXiv preprint arXiv:2411.19577*, 2024.

[111] T. Zohdinasab, V. Riccio, A. Gambi, and P. Tonella, "Efficient and effective feature space exploration for testing deep learning systems," *ACM Transactions Software Engineering and Methodology.*, vol. 32, no. 2, Mar. 2023. [Online]. Available: https://doi.org/10.1145/3544792

[112] I. G. Daza, R. Izquierdo, L. M. Martínez, O. Benderius, and D. F. Llorca, "Sim-to-real transfer and reality gap modeling in model predictive control for autonomous driving," *Applied Intelligence*, vol. 53, no. 10, pp. 12 719–12 735, 2023.

[113] Unity Technologies, "Unity real-time development platform," 2024, accessed: 2025-11-01. [Online]. Available: https://unity.com/

[114] NVIDIA Corporation, "Nvidia physx sdk," 2022, accessed: 2024-11-01. [Online]. Available: https://developer.nvidia.com/physx-sdk

[115] T. Huynh, A. Gambi, and G. Fraser, "Ac3r: Automatically reconstructing car crashes from police reports," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2019, pp. 31–34.

[116] V. Maddiralla and S. Subramanian, "Effective lane detection on complex roads with convolutional attention mechanism in autonomous vehicles," *Scientific Reports*, vol. 14, no. 1, p. 19193, 2024.

[117] G. Lian, "Robust lane detection based on informative feature pyramid network in complex scenarios," *Electronics*, vol. 14, no. 16, p. 3179, 2025.

[118] 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.

[119] 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 *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2018, pp. 132–142.

[120] M. Biagiola, A. Stocco, V. Riccio, and P. Tonella, "Two is better than one: digital siblings to improve autonomous driving testing," *Empirical Software Engineering*, vol. 29, no. 4, p. 72, 2024.

[121] A. Stocco, B. Pulfer, and P. Tonella, "Model vs system level testing of autonomous driving systems: a replication and extension study," *Empirical Software Engineering*, vol. 28, no. 3, p. 73, 2023.

[122] S. Söntges and M. Althoff, "Computing the drivable area of autonomous road vehicles in dynamic road scenes," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 6, pp. 1855–1866, 2017.

[123] ASAM e.V., "Opendrive format specification, rev. 1.4," Association for Standardization of Automation and Measuring Systems, Tech. Rep., 2015. [Online]. Available: https://www.asam.net/standards/detail/opendrive/

[124] S. Documentation, "Catmull-rom splines," 2024, accessed: 2024-11-19. [Online]. Available: https://splines.readthedocs.io/en/latest/euclidean/catmull-rom.html

[125] W. Zeng, T. Xiong, and C. Wang, "Optimization of smooth trajectories for two-wheel differential robots under kinematic constraints using clothoid curves," *Sensors*, vol. 25, no. 10, p. 3143, 2025.

[126] C. Yüksel, S. Schaefer, and J. Keyser, "On the parameterization of catmull-rom curves," *Computer-Aided Design*, vol. 43, no. 7, pp. 747–755, 2011.

[127] D. Shreiner, G. Sellers, J. Kessenich, and B. Licea-Kane, *OpenGL Programming Guide: The Official Guide to Learning OpenGL*, 8th ed. Addison-Wesley, 2013.

[128] Epic Games, "Unreal engine documentation," 2024, https://docs.unrealengine.com/.

[129] G. Farin, *Curves and Surfaces for CAGD: A Practical Guide*, 5th ed. Morgan Kaufmann, 2002.

[130] L. Piegl and W. Tiller, *The NURBS Book*. Springer, 1996.

[131] B. Schwab and T. H. Kolbe, "Validation of parametric opendrive road space models," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. X-4/W2-2022, pp. 257–264, 2022.

[132] Z. Qiao, Z. Yu, H. Yin, and S. Shen, "Online monocular lane mapping using catmull-rom spline," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 7179–7186.

[133] P. M. Johnson, H.-J. Kou, S. Agustin, C. Moore, J. Miglani, W. Doane, and D. Hale, "Beyond the personal software process: Metrics collection and analysis for software evolution," Collaborative Software Development Laboratory, University of Hawaii, Tech. Rep. CSDL-08-04, 2008. [Online]. Available: https://csdl.ics.hawaii.edu/techreports/2008/08-04/08-04.pdf

[134] Q. Ali, A. Stocco, L. Mariani, and O. Riganelli, "Coverage-guided road selection and prioritization for efficient testing in autonomous driving systems," *arXiv preprint arXiv:2601.08609*, 2026.

[135] P. S. Kochhar, T. F. Bissyandé, D. Lo, and L. Jiang, "Adoption of software testing in open source projects–a preliminary study on 50,000 projects," in *2013 17th European Conference on Software Maintenance and Reengineering*, 2013, pp. 353–356.

[136] R. M. d. Silva, C. Cruz, H. d. S. Campos, L. G. Murta, and V. d. O. Neves, "What is the adoption level of automated support for testing in open-source ecosystems?" in *Proceedings of the IV Brazilian Symposium on Systematic and Automated Software Testing*, 2019, pp. 80–89.

[137] M. Esposito, A. Janes, V. Lenarduzzi, and D. Taibi, "The invisible hand of ai libraries shaping open source projects and communities," *arXiv preprint arXiv:2601.01944*, 2026.

[138] C. Watson, N. Cooper, D. N. Palacio, K. Moran, and D. Poshyvanyk, "A systematic literature review on the use of deep learning in software engineering research," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 2, pp. 1–58, 2022.

[139] S. Albelali and M. Ahmed, "Testing machine learning and deep learning systems: Achievements and challenges," *Arabian Journal for Science and Engineering*, pp. 1–52, 2025.

[140] D. Patil, N. Rane, P. Desai, and J. Rane, "Machine learning and deep learning: Methods, techniques, applications, challenges, and future research opportunities," *Trustworthy artificial intelligence in industry and society*, pp. 28–81, 2024.

[141] A. Vogelsang and M. Borg, "Requirements engineering for machine learning: Perspectives from data scientists," in *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*. IEEE, 2019, pp. 245–251.

[142] A. Serban, K. van der Blom, H. Hoos, and J. Visser, "Software engineering practices for machine learning—adoption, effects, and team assessment," *Journal of Systems and Software*, vol. 209, p. 111907, 2024.

[143] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: A case study," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2019, pp. 291–300.

[144] L. Chen, P. Wu, K. Chitta, B. Jaeger, A. Geiger, and H. Li, "End-to-end autonomous driving: Challenges and frontiers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[145] D. Xie, D. Li, Y. Zhao, W. Liu, Y. Zhao, and N. Wang, "A survey on scenario-based test and evaluation methods for autonomous vehicles," *Proceedings of the*

Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering, p. 09544070251379541, 2025.

[146] C. Birchler, S. Khatiri, P. Rani, T. Kehrer, and S. Panichella, "A roadmap for simulation-based testing of autonomous cyber-physical systems: Challenges and future direction," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 5, pp. 1–9, 2025.

[147] HTEC Group. (2023, nov) 6 reasons why you should use simulation in autonomous vehicle software development. Accessed: 2026-01-27. [Online]. Available: https://htec.com/insights/blogs/6-reasons-why-you-should-use-simulation-in-autonomous-vehicle-software-development/

[148] C. Li, J. Sifakis, Q. Wang, R. Yan, and J. Zhang, "Simulation-based validation for autonomous driving systems," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023, pp. 842–853.

[149] Q. Ali, A. Stocco, L. Mariani, and O. Riganelli, "Opencat: Improving interoperability of ads testing," in *2025 IEEE/ACM International Workshop on Deep Learning for Testing and Testing for Deep Learning (DeepTest)*. IEEE, 2025, pp. 53–60, dataset available at https://github.com/lakhanqurban/OpenCat.

[150] E. Castellano, A. Cetinkaya, C. H. Thanh, S. Klikovits, X. Zhang, and P. Arcaini, "Frenetic at the SBST 2021 tool competition," in *14th IEEE/ACM International Workshop on Search-Based Software Testing, SBST 2021, Madrid, Spain, May 31, 2021*. IEEE, 2021, pp. 36–37.

[151] D. Humeniuk, G. Antoniol, and F. Khomh, "Ambiegen tool at the SBST 2022 tool competition," in *15th IEEE/ACM International Workshop on Search-Based Software Testing, SBST@ICSE 2022, Pittsburgh, PA, USA, May 9, 2022*. IEEE, 2022, pp. 43–46.

[152] E. Castellano, S. Klikovits, A. Cetinkaya, and P. Arcaini, "Freneticv at the SBST 2022 tool competition," in *15th IEEE/ACM International Workshop on Search-Based Software Testing, SBST@ICSE 2022, Pittsburgh, PA, USA, May 9, 2022*. IEEE, 2022, pp. 47–48.

[153] Y. Lin, M. Ratzel, and M. Althoff, "Automatic traffic scenario conversion from openscenario to commonroad," in *2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2023, pp. 4941–4946.

[154] N. Kochdumper, Y. Wang, J. Betz, and M. Althoff, "Results of the 2023 commonroad motion planning competition for autonomous vehicles," *arXiv preprint arXiv:2411.06425*, 2024.

[155] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," pp. 11 621–11 631, 2020.

[156] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine *et al.*, "Scalability in perception for autonomous driving: Waymo open dataset," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2446–2454.

[157] ASAM e.V., "Openscenario: A scenario description language for testing of automated driving functions," Association for Standardization of Automation and Measuring Systems, Tech. Rep., 2020. [Online]. Available: https://www.asam.net/standards/detail/openscenario/

[158] L. Baresi, D. Y. Xian Hu, A. Stocco, and P. Tonella, "Efficient domain augmentation for autonomous driving testing using diffusion models," in *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, 2025, pp. 398–410.

[159] M. Khatibsyarbini, M. A. Isa, D. N. Jawawi, and R. Tumeng, "Test case prioritization approaches in regression testing: A systematic literature review," *Information and Software Technology*, vol. 93, pp. 74–93, 2018.

[160] R. Pan, M. Bagherzadeh, T. A. Ghaleb, and L. Briand, "Test case selection and prioritization using machine learning: a systematic literature review," *Empirical Software Engineering*, vol. 27, no. 2, p. 29, 2022.

[161] Y. Jiang, S. Sun, and X. Zheng, "Regression testing optimization for ros-based autonomous systems: A comprehensive review of techniques," *arXiv preprint arXiv:2506.16101*, 2025.

[162] E. Castellano, A. Cetinkaya, and P. Arcaini, "Analysis of road representations in search-based testing of autonomous driving systems," in *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2021, pp. 167–178.

[163] J. Kerber, S. Wagner, K. Groh, D. Notz, T. Kühbeck, D. Watzenig, and A. Knoll, "Clustering of the scenario space for the assessment of automated driving," in *2020 IEEE Intelligent Vehicles Symposium (IV)*, 2020, pp. 578–583.

[164] C. Bernhard, A. Klem, E. C. Altuntas, and H. Hecht, "Wider is better but sharper is not: optimizing the image of camera-monitor systems," *Ergonomics*, vol. 65, no. 7, pp. 899–914, 2022.

[165] A. Piazzoni, J. Slavik, and J. Dauwels, “Vista: a framework for virtual scenario-based testing of automated vehicles,” in *2021 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2021, pp. 1540–1547.

[166] A. Güllü, F. A. Shah, and D. Pfahl, “An lstm-based test selection method for self-driving cars,” *arXiv preprint arXiv:2501.03881*, 2025.

[167] S. Riedmaier, T. Ponn, D. Ludwig, B. Schick, and F. Diermeyer, “Survey on scenario-based safety assessment of automated vehicles,” *IEEE access*, vol. 8, pp. 87 456–87 477, 2020.

[168] Hexagon AB, “Virtual test drive,” Hexagon.com, 2024, accessed: 2024-05-21. [Online]. Available: https://hexagon.com/products/virtual-test-drive

[169] TORCS Team, “The open racing car simulator,” 2014, open-source racing simulation software. [Online]. Available: https://sourceforge.net/projects/torcs/

[170] G. Rong, B. H. Shin, H. Tabatabaee, Q. Lu, S. Lemke, M. Možeiko, E. Boise, G. Uhm, M. Gerow, S. Mehta, E. Agafonov, T. H. Kim, E. Sterner, K. Ushiroda, M. Reyes, D. Zelenkovsky, and S. Kim, “Lgsvl simulator: A high fidelity simulator for autonomous driving,” in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, 2020, pp. 1–6.

[171] H. Zhao, M. Meng, X. Li, J. Xu, L. Li, and S. Galland, “A survey of autonomous driving frameworks and simulators,” *Advanced Engineering Informatics*, vol. 62, p. 102850, 2024.

[172] M. H. Amini and S. Nejati, “Bridging the gap between real-world and synthetic images for testing autonomous driving systems,” in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 2024, pp. 732–744.

[173] M. Althoff, M. Koschi, and S. Manzinger, “Commonroad: Composable benchmarks for motion planning on roads,” in *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 719–726.

[174] R. Queiroz, C. Berger, and J. Sjöberg, “From opendrive to lanelet2—a conversion tool for traffic scenario representation,” in *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2019, pp. 1906–1913.

[175] F. Poggenhans, J.-H. Pauls, J. Janosovits, S. Orf, M. Naumann, F. Kuhnt, and M. Mayr, “Lanelet2: A high-definition map framework for the future of automated driving,” in *2018 21st international conference on intelligent transportation systems (ITSC)*. IEEE, 2018, pp. 1672–1679.

[176] F. Bock, C. Sippl, A. Heinz, C. Lauer, and R. German, "Advantageous usage of textual domain-specific languages for scenario-driven development of automated driving functions," in *2019 IEEE International Systems Conference (SysCon)*, 2019, pp. 1–8.

[177] J.-W. Lin, N. Salehnamadi, and S. Malek, "Test automation in open-source android apps: A large-scale empirical study," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, 2020, pp. 1078–1089.

[178] J. Kerber, S. Wagner, K. Groh, D. Notz, T. Kühbeck, D. Watzenig, and A. Knoll, "Clustering of the scenario space for the assessment of automated driving," in *2020 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2020, pp. 578–583.

[179] C. Lu, H. Zhang, T. Yue, and S. Ali, "Search-based selection and prioritization of test scenarios for autonomous driving systems," in *International Symposium on Search Based Software Engineering*. Springer, 2021, pp. 41–55.

[180] I. Goodfellow, "Deep learning," 2016.

[181] S. Sultana and B. Ahmed, "Lane detection and tracking under rainy weather challenges," in *2021 IEEE Region 10 Symposium (TENSYMP)*. IEEE, 2021, pp. 1–6.

[182] Z. Rahman and B. T. Morris, "Lvlane: deep learning for lane detection and classification in challenging conditions," in *2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2023, pp. 3901–3907.

[183] K. Ahuja, Y. Tian, J. Chen, and M. Kim, "Testing deep learning models: A first comparative study of four test adequacy criteria," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2022, pp. 1–12.

[184] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "{TensorFlow}: a system for {Large-Scale} machine learning," in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, 2016, pp. 265–283.

[185] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[186] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[187] PyPI, "Pytest," https://docs.pytest.org/en/8.0.x/, 2024.

[188] K. Beck, "unittest," https://docs.python.org/3/library/unittest.html, 2024.

[189] Travis CI. (2026) Travis ci: Simple, flexible, trustworthy ci/cd tools. Travis CI. Accessed: 2026-01-27. [Online]. Available: https://www.travis-ci.com/

[190] S. Raschka, J. Patterson, and C. Nolet, "Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence," *Information*, vol. 11, no. 4, p. 193, 2020.

[191] T. Preston-Werner, "Github search api," https://docs.github.com/en/rest/quickstart?apiVersion=2022-11-28, 2024.

[192] M. M. YAPICI and N. Topaloğlu, "Performance comparison of deep learning frameworks," *Computers and Informatics*, vol. 1, no. 1, pp. 1–11, 2021.

[193] S. Madhavan. Compare deep learning frameworks. [Online]. Available: https://developer.ibm.com/articles/compare-deep-learning-frameworks

[194] V. Meel. Top 10 deep learning frameworks in 2024. [Online]. Available: https://viso.ai/deep-learning/deep-learning-frameworks/

[195] H. Borges, A. Hora, and M. T. Valente, "What is the impact of social attributes on github projects?" in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1. IEEE, 2016, pp. 345–355.

[196] L. M. Qurban Ali, Oliviero Riganelli, "Material," https://github.com/lakhanqurban/PDLTesting, 2024.

[197] A. Leitner, I. Ciupa, B. Meyer, and M. Howard, "Reconciling manual and automated testing: The autotest experience," in *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*. IEEE, 2007, pp. 261a–261a.

[198] K. Beck, "Junit," https://junit.org/junit5/, 2024.

[199] C. Beust, "Testng," https://testng.org//, 2024.

[200] C. Nakazawa, "Jest," https://jestjs.io/, 2024.

[201] Eich, "Mocha," https://mochajs.org/, 2024.

[202] I. Google, "Googletest," https://github.com/google/googletest, 2024.

[203] S. Madhavan, "Xunit," https://xunit.net, 2024.

[204] M. Poliarush, "awesome-test-automation." GitHub, 2015. [Online]. Available: https://github.com/atinfo/awesome-test-automation/

[205] Future Skills Academy. (2024) CI/CD for Machine Learning. Blog Post. [Online]. Available: https://futureskillsacademy.com/blog/ci-cd-for-machine-learning/

[206] Q. Yang, J. J. Li, and D. Weiss, "A survey of coverage based testing tools," ser. AST '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 99–103. [Online]. Available: https://doi.org/10.1145/1138929.1138949

[207] J. H. Andrews, L. C. Briand, Y. Labiche, and A. S. Namin, "Using mutation analysis for assessing and comparing testing coverage criteria," *IEEE Transactions on Software Engineering*, vol. 32, no. 8, pp. 608–624, 2006.

[208] P. S. Kochhar, D. Lo, J. Lawall, and N. Nagappan, "Code coverage and postrelease defects: A large-scale study on open source projects," *IEEE Transactions on Reliability*, vol. 66, no. 4, pp. 1213–1228, 2017.

[209] M. Yaseen, A. Mustapha, and N. Ibrahim, "Prioritization of software functional requirements from developers perspective," *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 9, 2020. [Online]. Available: http://dx.doi.org/10.14569/IJACSA.2020.0110925

[210] C. R. Camacho, S. Marczak, and D. S. Cruzes, "Agile team members perceptions on non-functional testing: influencing factors from an empirical study," in *2016 11th international conference on availability, reliability and security (ARES)*. IEEE, 2016, pp. 582–589.

[211] J. Engelberg and E. Hooten, "Codecov," https://about.codecov.io/, 2024.

[212] I. Coveralls, "Coveralls," https://coveralls.io/, year=2024.

[213] N. C. Zakas, "Scrutinizer," https://scrutinizer-ci.com/, 2024.

[214] L. Inozemtseva and R. Holmes, "Coverage is not strongly correlated with test suite effectiveness," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 435–445. [Online]. Available: https://doi.org/10.1145/2568225.2568271

[215] 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)*, 2019, pp. 1039–1049.

[216] X. Sun, R. Cheng, J. Chen, E. Ang, O. Legunsen, and T. Xu, "Testing configuration changes in context to prevent production failures," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 735–751.

[217] H. Guo, C. Tao, Z. Huang, and W. Zou, "Coverage-guided testing for deep learning models: A comprehensive survey," *arXiv preprint arXiv:2507.00496*, 2025.

[218] E. Panourgia, T. Plessas, I. Balampanis, and D. Spinellis, "Good tools are half the work: Tool usage in deep learning projects," *arXiv preprint arXiv:2310.19124*, 2023.

[219] T. Zhang, H. Liu, W. Wang, and X. Wang, "Virtual tools for testing autonomous driving: A survey and benchmark of simulators, datasets, and competitions," *Electronics*, vol. 13, no. 17, p. 3486, 2024.

[220] N. Neelofar and A. Aleti, "Identifying and explaining safety-critical scenarios for autonomous vehicles via key features," *ACM Transactions Software Engineering and Methodology.*, vol. 33, no. 4, Apr. 2024.

[221] ——, "Towards Reliable AI: Adequacy Metrics for Ensuring the Quality of System-level Testing of Autonomous Vehicles," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE).* ACM, 2024.

[222] V. Crespo-Rodriguez, Neelofar, and A. Aleti, "PAFOT: A Position-Based Approach for Finding Optimal Tests of Autonomous Vehicles," in *Proceedings of the ACM/IEEE 5th International Conference on Automation of Software Test (AST 2024)*, ser. AST '24. ACM, 2024, p. 159–170.

[223] C. Lu, S. Ali, and T. Yue, "Epitester: Testing autonomous vehicles with epigenetic algorithm and attention mechanism," *IEEE Transactions on Software Engineering*, pp. 1–19, 2024.

[224] C. Lu, T. Yue, M. Zhang, and S. Ali, "DeepQTest: Testing Autonomous Driving Systems with Reinforcement Learning and Real-world Weather Data," 2023. [Online]. Available: https://arxiv.org/abs/2310.05170

[225] Q. Pan, T. Wang, P. Arcaini, T. Yue, and S. Ali, "Safety assessment of vehicle characteristics variations in autonomous driving systems," 2023. [Online]. Available: https://arxiv.org/abs/2311.14461

[226] F. Klück, Y. Li, J. Tao, and F. Wotawa, "An empirical comparison of combinatorial testing and search-based testing in the context of automated and autonomous driving systems," *Information and Software Technology*, vol. 160, p. 107225, 2023.

[227] F. Klück, D. Sumann, and F. Wotawa, "Utilizing genetic algorithms for generating critical scenarios for testing autonomous driving functions," in *2024 IEEE International Conference on Artificial Intelligence Testing (AITest)*, 2024, pp. 73–80.

[228] Z. Tu, L. Niu, W. Fan, and T. Zhang, "Multi-modal traffic scenario generation for autonomous driving system testing," *Proceedings of the ACM on Software Engineering*, vol. 2, no. FSE, pp. 1733–1756, 2025.

[229] P. Arcaini and A. Cetinkaya, "Crag–a combinatorial testing-based generator of road geometries for ads testing," *Science of Computer Programming*, vol. 238, p. 103171, 2024.

[230] S. Klikovits, V. Riccio, E. Castellano, A. Cetinkaya, A. Gambi, and P. Arcaini, "Does road diversity really matter in testing automated driving systems?–a registered report," *arXiv preprint arXiv:2209.05947*, 2022.

[231] Y. Wang, A. Alhuraish, S. Yuan, and H. Zhou, "Openlka: An open dataset of lane keeping assist from recent car models under real-world driving conditions," *arXiv preprint arXiv:2505.09092*, 2025.

[232] O. Zendel, M. Murschitz, M. Humenberger, and W. Herzner, "Unreal stereo: A synthetic dataset for analyzing stereo vision," in *German Conference on Pattern Recognition (GCPR)*. Springer, 2019, pp. 40–55.

[233] E. E. Catmull and R. Rom, "A class of local interpolating splines," *Computer Aided Geometric Design*, pp. 317–326, 1974. [Online]. Available: https://api.semanticscholar.org/CorpusID:118383557

[234] Association for Standardization of Automation and Measuring Systems (ASAM), "ASAM OpenDRIVE®," https://www.asam.net/standards/detail/opendrive/, 2023, accessed: 2024-11-04.

[235] American Association of State Highway and Transportation Officials, *A Policy on Geometric Design of Highways and Streets*, 7th ed. Washington, DC: American Association of State Highway and Transportation Officials, 2018.

[236] M. Pagel *et al.*, "libopendrive," 2024, accessed: 2024-11-04. [Online]. Available: https://github.com/pageldev/libOpenDRIVE

[237] S. Pagel, "Opendrive viewer," https://sebastian-pagel.net/odrviewer/, accessed: 2024-11-04.

[238] S. Wright, "The method of path coefficients," *The Annals of Mathematical Statistics*, vol. 5, no. 3, pp. 161–215, 1934. [Online]. Available: http://www.jstor.org/stable/2957502

[239] D. Chicco, M. J. Warrens, and G. Jurman, "The coefficient of determination r-squared is more informative than smape, mae, mape, mse and rmse in regression analysis evaluation," *Peerj computer science*, vol. 7, p. e623, 2021.

[240] M. Alsherif, M. Daowd, A. Bassiuny, and H. A. Metered, "Utilizing transfer learning in the udacity simulator to train a self-driving car for steering angle prediction," in

2023 Eleventh International Conference on Intelligent Computing and Information Systems (ICICIS). IEEE, 2023, pp. 134–139.

[241] L. Sorokin, M. Biagiola, and A. Stocco, "Simulator ensembles for trustworthy autonomous driving testing," *arXiv preprint arXiv:2503.08936*, 2025.

[242] C. Birchler, T. K. Mohammed, P. Rani, T. Nechita, T. Kehrer, and S. Panichella, "How does simulation-based testing for self-driving cars match human perception?" *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 929–950, 2024.

[243] F. Yang, X. Li, Q. Liu, X. Li, and Z. Li, "Learning-based hierarchical decision-making framework for automatic driving in incompletely connected traffic scenarios," *Sensors*, vol. 24, no. 8, p. 2592, 2024.

[244] J. Fernandez, "Test case generation for lane-keeping functions in autonomous vehicles using search-based software engineering techniques," *National Library Library Network Services*, 2024.

[245] BeamNG GmbH, "Ai and traffic," https://documentation.beamng.com/tutorials/ai/, 2025, last modified: September 5, 2025. Accessed: December 13, 2025.

[246] J. H. Rife, P. Elwood, and H. Wassaf, "Event-based risk assessment for alert limits in automotive lane keeping," in *2023 IEEE/ION Position, Location and Navigation Symposium (PLANS)*. IEEE, 2023, pp. 621–629.

[247] V. Surblys, V. Žuraulis, and T. Tinginys, "Research on reliability of vehicle line detection and lane keeping systems," *Sustainability*, vol. 17, no. 22, p. 10222, 2025.

[248] L. Zhang, D. Marinov, L. Zhang, and S. Khurshid, "An empirical study of JUnit test-suite reduction," in *Proceedings of the 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2013, pp. 398–407.

[249] M. Fowler and M. Foemmel, "Continuous integration," *ThoughtWorks*, pp. 122–127, 2006. [Online]. Available: https://www.martinfowler.com/articles/originalContinuousIntegration.html

[250] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Softw. Test. Verif. Reliab.*, vol. 22, no. 2, p. 67–120, Mar. 2012. [Online]. Available: https://doi.org/10.1002/stv.430

[251] V. Riccio, G. Jahangirova, A. Stocco, N. Humbatova, M. Weiss, and P. Tonella, "Testing Machine Learning based Systems: A Systematic Mapping," *Empirical Software Engineering*, 2020.

[252] S. Coropulis, N. Berloco, P. Intini, and V. Ranieri, "Road design influence on driving behaviors: the influence of curve design, a case study," *Transport Economics and Management*, vol. 3, pp. 104–116, 2025.

[253] J. Gielis, "A generic geometric transformation that unifies a wide range of natural and abstract shapes," *American Journal of Botany*, vol. 90, no. 3, pp. 333–338, 2003.

[254] X. Zhao, D. Lord, and Y. Peng, "Examining network segmentation for traffic safety analysis with data-driven spectral analysis," *IEEE Access*, vol. 7, pp. 120 744–120 757, 2019.

[255] S. Cafiso and A. Di Graziano, "Definition of homogenous sections in road pavement measurements," *Procedia-Social and Behavioral Sciences*, vol. 53, pp. 1069–1079, 2012.

[256] J. Gielis, "A generic geometric transformation that unifies a wide range of natural and abstract shapes," *American Journal of Botany*, vol. 90, no. 3, pp. 333–338, 2003.

[257] P. Malgouyres and F. Brunet, "Digital curvature estimation based on osculating circle," *Machine Vision and Applications*, vol. 18, no. 3-4, pp. 229–251, 2007.

[258] S. Khan, D.-H. Lee, M. A. Khan, A. R. Gilal, J. Iqbal, and A. Waqas, "Efficient and improved edge detection via a hysteresis thresholding method," *Current Science*, vol. 118, no. 6, pp. 954–960, 2020.

[259] M. Qaseem Ghadi and Á. Török, "Comparison of different road segmentation methods," *Promet-Traffic&Transportation*, vol. 31, no. 2, pp. 163–172, 2019.

[260] Y. Liu, Y. Wang, H. Zhang, and Q. Li, "Geometric constraints and semantic optimization slam algorithm for dynamic scenarios," *Scientific Reports*, vol. 15, no. 1, p. 31864, 2025.

[261] G. Jahangirova, A. Stocco, and P. Tonella, "Quality metrics and oracles for autonomous vehicles testing," in *Proceedings of the IEEE 14th International Conference on Software Testing, Verification and Validation*, ser. ICST '21. IEEE, 2021.

[262] M. Biagiola and P. Tonella, "Boundary state generation for testing and improvement of autonomous driving systems," *IEEE Transactions on Software Engineering*, vol. 50, no. 8, p. 2040–2053, Jul. 2024. [Online]. Available: https://doi.org/10.1109/TSE.2024.3420816

[263] F. U. Haq, D. Shin, S. Nejati, and L. Briand, "Can offline testing of deep neural networks replace their online testing? a case study of automated driving systems," *Empirical Software Engineering*, vol. 26, no. 5, p. 90, 2021.

[264] R. Rajamani, *Vehicle Dynamics and Control*, 2nd ed. Springer Science & Business Media, 2012.

[265] J. Y. Wong, *Theory of Ground Vehicles*, 4th ed. John Wiley & Sons, 2008.

[266] E. Eskin, "A geometric framework for unsupervised anormaly detection: Detecting intrusions in unlabeled data," *Applications of Data Mining in Computer Security*, 2002.

[267] F. Murtagh and P. Contreras, "Algorithms for hierarchical clustering: an overview," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, no. 1, pp. 86–97, 2012.

[268] S. C. Johnson, "Hierarchical clustering schemes," *Psychometrika*, vol. 32, no. 3, pp. 241–254, 1967.

[269] S. Sharma, N. Batra *et al.*, "Comparative study of single linkage, complete linkage, and ward method of agglomerative clustering," in *2019 international conference on machine learning, big data, cloud and parallel computing (COMITCon)*. IEEE, 2019, pp. 568–573.

[270] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2007, pp. 1027–1035.

[271] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651–666, 2010.

[272] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 1996, pp. 226–231.

[273] E. G. Cartaxo, P. D. Machado, and F. G. O. Neto, "On the use of a similarity function for test case selection in the context of model-based testing," *Software Testing, Verification and Reliability*, vol. 21, no. 2, pp. 75–100, 2011.

[274] R. Sibson, "SLINK: An optimally efficient algorithm for the single-link cluster method," *The Computer Journal*, vol. 16, no. 1, pp. 30–34, 1973.

[275] G. E. Blelloch, R. Peng, and K. Tangwongsan, "Linear-work greedy parallel approximate set cover and variants," in *Proceedings of the twenty-third annual ACM symposium on Parallelism in algorithms and architectures*, 2011, pp. 23–32.

[276] T. Laurent, S. Klikovits, P. Arcaini, F. Ishikawa, and A. Ventresque, "Parameter coverage for testing of autonomous driving systems under uncertainty," *ACM Transactions Software Engineering and Methodology.*, vol. 32, no. 3, Apr. 2023.

[277] L. Zhang, D. Marinov, L. Zhang, and S. Khurshid, "An empirical study of junit test-suite reduction," in *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2013, pp. 398–407.

[278] T. B. Noor and H. Hemmati, "A similarity-based approach for test case prioritization using historical failure data," in *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2015, pp. 58–68.

[279] I. Chang, J. Lee, and S. Ahn, "Evaluation of level 2 automated driving safety on curved sections," *KSCE Journal of Civil Engineering*, vol. 28, no. 6, 2024.

[280] T. Zohdinasab, V. Riccio, and P. Tonella, "Focused test generation for autonomous driving systems," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 33, no. 6, pp. 1–32, 2024.

[281] X. Ji, L. Xue, Z. He, and X. Luo, "Autonomous driving system testing via diversity-oriented driving scenario exploration," *ACM Transactions on Software Engineering and Methodology*, 2025.

[282] J.-M. Kim and A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments," in *Proceedings of the 24th International Conference on Software Engineering (ICSE)*. ACM/IEEE, 2002, pp. 119–129.

[283] U. D. of Transportation, "A framework for automated driving system testable cases and scenarios," https://rosap.ntl.bts.gov/view/dot/38824/dot_38824_DS1.pdf, 2018.

[284] ——, "Standing general order on crash reporting for level 2 advanced driver assistance systems," https://www.nhtsa.gov/sites/nhtsa.gov/files/2022-06/ADAS-L2-SGO-Report-June-2022.pdf, 2022.

[285] J. Opletal. (2025, Jul.) China's massive adas test: 36 cars, 15 hazard scenarios, 216 crashes. [Online]. Available: https://carnewschina.com/2025/07/24/chinas-massive-adas-test-36-cars-15-hazard-scenarios-216-crashes/

[286] Baidu Inc., "Baidu Apolloscapes Dataset," https://apolloscape.auto/index.html, 2018, accessed: [2024-01-15].

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

[288] 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 IEEE/ACM 33rd ASE*, ser. ASE '18. ACM, 2018, p. 132–142.

[289] D. Saad, "Online algorithms and stochastic approximations," *Online Learning*, 1998.

[290] Udacity, "Udacity self-driving car simulator," https://github.com/udacity/self-driving-car-sim, 2021, accessed: [2026-01-15].

[291] F. U. Haq, D. Shin, S. Nejati, and L. Briand, "Comparing offline and online testing of deep neural networks: An autonomous car case study," in *Proceedings of 13th IEEE International Conference on Software Testing, Verification and Validation*, ser. ICST '20. IEEE, 2020.

[292] A. Stocco, B. Pulfer, and P. Tonella, "Model vs system level testing of autonomous driving systems: a replication and extension study," *Empirical Software Engineering*, vol. 28, no. 3, p. 73, May 2023.

[293] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimisation: NSGA-II," in *Parallel Problem Solving from Nature — PPSN VI*, ser. Lecture Notes in Computer Science, vol. 1917. Springer, 2000, pp. 849–858.

[294] S. Panichella, A. Gambi, F. Zampetti, and V. Riccio, "SBST tool competition 2021," in *2021 IEEE/ACM 14th International Workshop on Search-Based Software Testing (SBST)*. IEEE, 2021, pp. 20–27.

[295] E. Castellano, S. Klikovits, A. Cetinkaya, and P. Arcaini, "Freneticv at the SBST 2022 tool competition," in *2022 IEEE/ACM 15th International Workshop on Search-Based Software Testing (SBST)*. IEEE, 2022, pp. 47–48.

[296] A. Gambi, G. Jahangirova, V. Riccio, and F. Zampetti, "SBST tool competition 2022," in *Proceedings of the 15th IEEE/ACM International Workshop on Search-Based Software Testing*, ser. SBST@ICSE 2022. IEEE, 2022, pp. 25–32.

[297] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Test case prioritization: an empirical study," *Proceedings IEEE International Conference on Software Maintenance - 1999 (ICSM'99). 'Software Maintenance for Business Change' (Cat. No.99CB36360)*, pp. 179–188, 1999. [Online]. Available: https://api.semanticscholar.org/CorpusID:15621196

[298] P. Jaccard, "étude comparative de la distribution florale," *Bulletin de la Société Vaudoise des Sciences Naturelles*, vol. 37, pp. 547–579, 1901.

[299] H. M. Eraqi, M. N. Moustafa, and J. Honer, "End-to-end deep learning for steering autonomous vehicles considering temporal dependencies," *arXiv preprint arXiv:1704.03004*, 2017.

[300] ——, "End-to-end deep learning for steering autonomous," in *OpenReview*, 2017.

[301] Z. Zhao, L. Alzubaidi, J. Zhang, Y. Duan, and Y. Gu, "A comparison review of transfer learning and self-supervised learning: Definitions, applications, advantages and limitations," *Expert Systems with Applications*, vol. 242, p. 122807, 2024.

[302] S. C. Lambertenghi, M. F. Valdez, and A. Stocco, "A multi-modality evaluation of the reality gap in autonomous driving systems," *arXiv preprint arXiv:2509.22379*, 2025.

[303] D. J. Fremont, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, and S. A. Seshia, "Formal scenario-based testing of autonomous vehicles: From simulation to the real world," in *IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2020, pp. 1–8.

Doctoral thesis carried out within the framework of the ISCS project funded by the PNRR Mission 4 Component 2 Investment 1.4, funded by the European Union – NextGenerationEU – CUP H43C22000520001