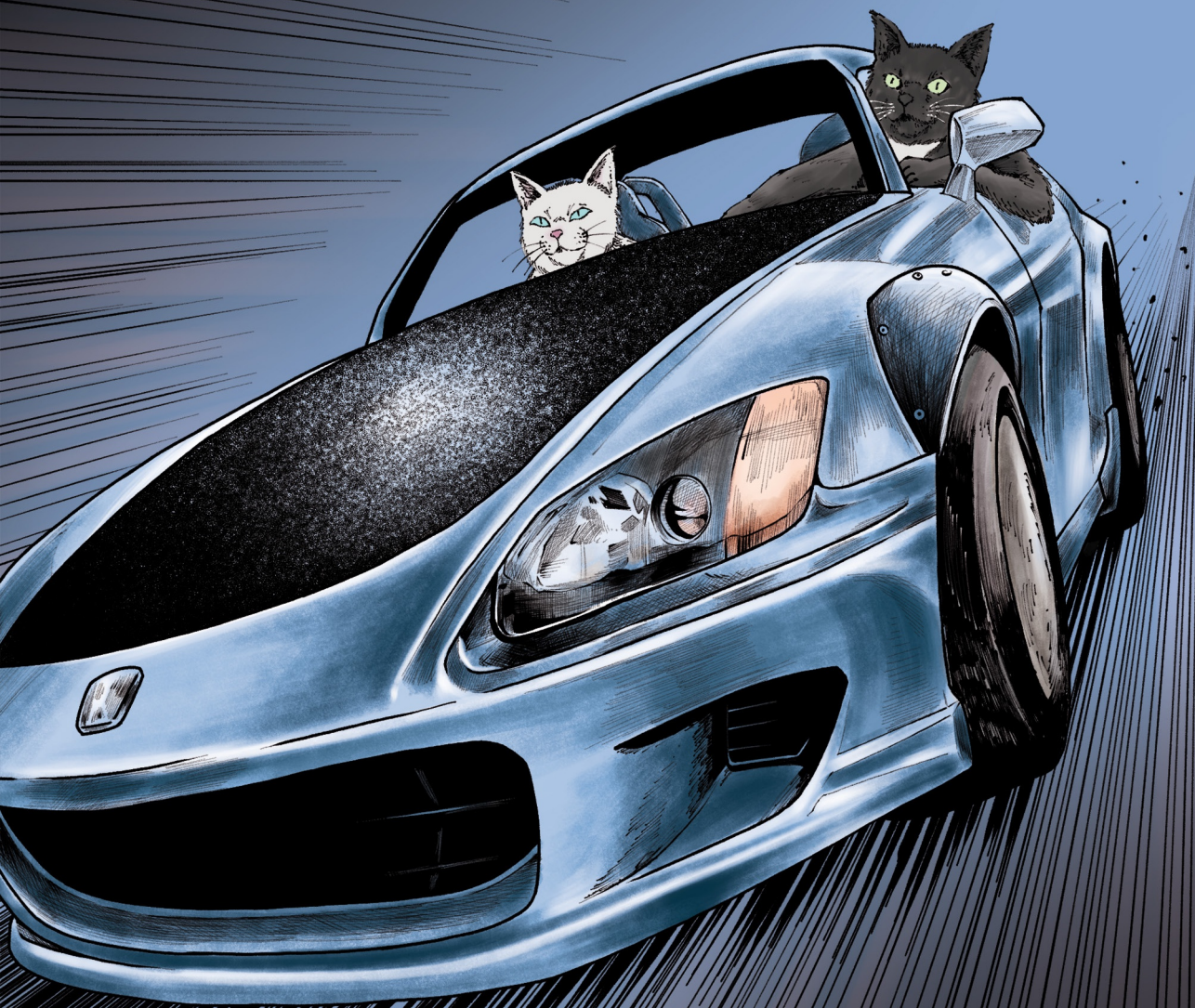
THE GOOD, THE BAD, AND THE ADVERSARIAL:

MULTI-OBJECTIVE TESTING FOR DEEP LEARNING SYSTEMS



ANTONY BARTLETT

Propositions

accompanying the dissertation

The Good, the Bad, and the Adversarial:

Multi-Objective Testing for Deep Learning Systems

by

Antony Bartlett

1. Education must prioritize manual programming proficiency before AI assistance to preserve both rigor and innovation.
2. Industry and academia benefit most from establishing hybrid work as the standard.
3. Academic research should be treated like open-source software.
4. Industrial animal agriculture is a greater impediment to achieving global climate goals than the entire transportation sector.
5. Without proper diversity, testing practices would be more likely to fall into local optima. (This proposition pertains to Chapter 4 of this thesis.)
6. We lost the em dash, but AI brought it back, we should once again embrace the em dash.
7. Machine Learning is here to stay, we should understand how to properly integrate it into our workflows, rather than attempting to avoid it.
8. Python is a great language for prototyping, but a terrible language for production code.
9. The EUs proposed ban on the term 'meat' in plant-based alternatives is misguided and will have negative consequences for both consumers and producers.
10. Given the correct stopping criteria, multi-objective methods can outperform single-objective methods in both speed and quality of results. (This proposition pertains to Chapter 3 and Chapter 4 of this thesis.)

These propositions are regarded as opposable and defendable, and have been approved as such by the promotors dr. ir. C. C. S. Liem and dr. A. Panichella.

The Good, the Bad, and the Adversarial:

Multi-Objective Testing for Deep Learning Systems

The Good, the Bad, and the Adversarial:

Multi-Objective Testing for Deep Learning Systems

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus,
Prof. dr. ir. H. Bijl,
Chair of the Board for Doctorates
to be defended publicly on
Tuesday, 27 October 2026 at 12:30

by

Antony Bartlett

This dissertation has been approved by the promotors.

Composition of the doctoral committee:

Rector Magnificus,	chairperson
Dr. ir. C.C.S. Liem	Delft University of Technology, *promotor*
Dr. A. Panichella	Delft University of Technology, *promotor*

Independent members:

Prof. dr. ir. B. van Arem	Delft University of Technology
Prof. dr. ir. J. Hellendoorn	Delft University of Technology
Dr. A. Arrieta	Mondragon University
Dr. G. Jahangirova	King's College London
Dr. B. Özkan	Delft University of Technology



Keywords: Search-based Software Engineering, Deep Neural Networks, Adversarial Examples, Software Testing
Printed by: ProefschriftMaken
Cover by: Antony Bartlett

Copyright © 2026 by Antony Bartlett

The author set this thesis using Quarto: https://github.com/quarto-tudelft.

ISBN 978-94-6563-046-5

An electronic copy of this dissertation is available at
https://repository.tudelft.nl/.

To my partner, Kelley, for all your amazing love, food, and support, and to my mum, Helen, for my ZX81–the computer that started it all for me.

Praise the sun!

Knight Solaire, Dark Souls

TABLE OF CONTENTS

Summary xi

Samenvatting xiii

Acknowledgements xv

1. Introduction 1
- 1.1. Background . . . 2
  - 1.1.1. Search-Based Software Testing . . . 3
  - 1.1.2. Computer Vision . . . 4
  - 1.1.3. Deep, Deep and Deeper . . . 5
  - 1.1.4. Simulation Environments . . . 7
  - 1.1.5. Large Language Models . . . 7
- 1.2. Challenges . . . 8
  - 1.2.1. The Black Box Nature of AI-Intensive Systems . . . 8
  - 1.2.2. Simulation Environments and Validation Runtime . . . 9
  - 1.2.3. High-Dimensional Search Spaces and SBST . . . 9
  - 1.2.4. Test Case Diversity . . . 10
- 1.3. Research Goals & Questions . . . 11
- 1.4. Research Methodology . . . 12
  - 1.4.1. Research Principles . . . 13
- 1.5. Research Outline . . . 14
- 1.6. Origins of Chapters . . . 17
- 1.7. Open Science . . . 18
  - 1.7.1. Open Data . . . 19
  - 1.7.2. Open Software . . . 19
- 1.8. Other Contributions . . . 19

I. Research Chapters 21

2. Multi-Objective Differential Evolution In The Generation Of Adversarial Examples 23
- 2.1. Introduction . . . 24
- 2.2. Related Work . . . 26
- 2.3. Approach . . . 27
  - 2.3.1. Single and Multi-objective Differential Evolution . . . 29

2.3.2. Post Processing . . . 33
2.4. Study Design . . . 33
2.4.1. Dataset . . . 34
2.4.2. Implementation and Parameter settings . . . 35
2.4.3. Evaluation Criteria . . . 36
2.5. Empirical Evaluation . . . 37
2.5.1. Results for RQ1 . . . 37
2.5.2. Results for RQ2 . . . 41
2.6. Threats to validity . . . 44
2.6.1. Construct validity . . . 44
2.6.2. Internal validity . . . 44
2.6.3. External validity . . . 45
2.6.4. Conclusion validity . . . 45
2.7. Conclusion and Future Work . . . 45

3. The Pursuit of Diversity: Multi-Objective Testing of Deep Reinforcement Learning Agents 47
3.1. Introduction . . . 47
3.2. Background and Related Work . . . 49
3.2.1. Deep Reinforcement Learning . . . 49
3.2.2. Black-box Testing . . . 50
3.2.3. Surrogates . . . 50
3.2.4. Testing DRL Agents . . . 50
3.3. Approach . . . 51
3.3.1. Surrogate models . . . 52
3.3.2. Diversity Metrics . . . 52
3.3.3. Multi-Objective Evolutionary Testing . . . 53
3.4. Empirical Evaluation . . . 54
3.4.1. Case Studies . . . 54
3.4.2. Baseline . . . 55
3.4.3. Implementation and Parameter Settings . . . 55
3.4.4. Evaluation Criteria . . . 55
3.4.5. Failure Detection and Diversity Assessment . . . 56
3.5. Results . . . 56
3.5.1. Configuration Comparison (RQ1) . . . 58
3.5.2. Comparison with INDAGO (RQ2) . . . 58
3.6. Threats To Validity . . . 60
3.7. Conclusion and Future Work . . . 60

4. Snakes On a 2-D Plane: Geometry-Based Image Representations for Test Case Selection in Autonomous Driving Simulation 63
4.1. Introduction . . . 64
4.2. Background and Related Work . . . 66
4.2.1. Test Case Selection for SDC Simulation Testing . . . 66
4.2.2. Simulation-based Testing for Self-driving Cars . . . 67
4.2.3. Learning-based Test Case Selection for SDCs . . . 67

4.3. Approach . . . 68
4.3.1. Training Data Generation . . . 68
4.3.2. Convolutional Neural Network Architectures . . . 74
4.3.3. Parameter Optimization . . . 76
4.3.4. Test Selection . . . 77
4.4. Empirical Evaluation . . . 80
4.4.1. Dataset . . . 80
4.4.2. Baseline . . . 81
4.4.3. Implementation and Parameter Settings . . . 81
4.4.4. Evaluation Procedure . . . 81
4.5. Results . . . 82
4.5.1. RQ1 : Dataset Choice Across CNN Architectures . . . 82
4.5.2. RQ2 : CNN Performance Against Existing Test Selection Techniques . . . 85
4.6. Threats To Validity . . . 87
4.6.1. Internal Validity . . . 87
4.6.2. External Validity . . . 88
4.6.3. Construct Validity . . . 89
4.6.4. Conclusion Validity . . . 90
4.7. Conclusion and Future Work . . . 90

5. The Last Dependency Crusade: Solving Python Dependency Conflicts with LLMs 93
5.1. Introduction . . . 94
5.2. Background and Related Work . . . 95
5.2.1. Existing Approaches . . . 95
5.2.2. Datasets of Python Dependency Conflicts . . . 96
5.3. Our Approach . . . 96
5.3.1. Inferring Module Names and Python Version (Stage A) . . . 97
5.3.2. Inferring Module Versions (Stage B) . . . 97
5.3.3. Docker-based Build and Validation (Stage C) . . . 98
5.3.4. Error Analysis and Classification (Stage D) . . . 98
5.3.5. Feedback Loop and Iterative Refinement (Stage E) . . . 99
5.3.6. Parallel Multi-Version Execution . . . 100
5.4. Experimental Setup . . . 100
5.4.1. LLM Model Selection . . . 100
5.4.2. Dataset . . . 101
5.4.3. Implementation and Parameter settings . . . 101
5.4.4. Evaluation Criteria . . . 102
5.5. Empirical Results . . . 103
5.5.1. Results for RQ1 . . . 103
5.5.2. Results for RQ2 . . . 105
5.5.3. Results for RQ3 . . . 106
5.6. Threats to validity . . . 108
5.6.1. Construct validity . . . 108

5.6.2. Internal validity 108
5.6.3. External validity 109
5.6.4. Conclusion validity 109
5.7. Conclusion and Future Work 110

6. Conclusion **111**
6.1. Research Questions Revisited 112
6.1.1. The Challenge of Minimal Perturbations in AI Testing 112
6.1.2. The Critical Role of Diversity in AI Testing 113
6.1.3. Surrogate Models in Adversarial Example Generation and Test Case Selection 114
6.1.4. The Reproducibility Challenge in AI Testing 115
6.2. Contributions to Future Impact 116
6.3. Future Work 117
6.3.1. Improved Search Strategies 117
6.3.2. Multi-Agent Testing 118
6.3.3. Large Scale Ground-Truthed Datasets 118
6.3.4. Humans Are Chaos Agents 119

Bibliography **121**

Glossary **139**

Curriculum Vitae **143**

List Of Publications **147**

SUMMARY

Artificial Intelligence (AI)-intensive systems, particularly Computer Vision (CV) models and Deep Reinforcement Learning (DRL) agents, are increasingly deployed in real-world safety-critical systems, such as autonomous vehicles and medical diagnosis. Their inclusion in these domains necessitates rigorous testing to ensure reliability and safety, particularly as we lack clear test oracles, face input fragility, must navigate vast and complex test spaces, and contend with dynamic and stochastic behaviors that defy traditional testing approaches.

In this thesis, we employ a purely black-box testing strategy to more accurately reflect real-world deployment scenarios, where model internals are inaccessible. We develop novel testing methodologies that combine proven search-based software testing (SBST) techniques with approaches harnessing surrogate models and Large Language Models (LLMs). We begin by presenting a pure differential evolution approach for generating minimal adversarial perturbations, extending it to multi-objective testing methods that enable trade-offs between attack effectiveness and imperceptibility to expose a broader range of input vulnerabilities. Building on this foundation, we introduce a multi-objective testing framework for Deep Reinforcement Learning that jointly optimizes for failure detection and test scenario diversity while leveraging lightweight surrogate models; ensuring that discovered faults span a broader spectrum of agent behaviors rather than exploiting only certain parts of a system. We then investigate the use of geometry-based image representations to guide test case selection in autonomous driving test suites, combining convolutional neural networks (CNNs) with optimization techniques for test case selection and prioritization. Finally, we investigate the use of LLMs to automatically resolve build errors in a reproducible testing framework, enabling the correct testing of AI-intensive systems without labor-intensive manual debugging.

The results of our experiments demonstrate that approaches that optimize for multiple objectives are more effective at discovering issues across a broader range of scenarios, while remaining computationally efficient. More importantly, we demonstrate that surrogate models can effectively enhance SBST techniques by approximating the behavior of the original system under test. Surrogate models provide an efficient fitness function based on their predictions to guide the search process, as surrogate models closely approximate model output without expensive evaluations on the original system. Finally, we demonstrate that LLMs can support the testing process by leveraging natural language processing to identify and resolve system build errors, thereby enabling reproducible system validation.

In summary, this thesis contributes novel hybrid search-based approaches for testing AI systems, addressing the challenges inherent to testing CV models and DRL agents. We demonstrate the effectiveness of combining surrogate models and LLMs with existing SBST methodologies, and highlight the critical importance of diversity in generating comprehensive test suites for robust AI systems.

SAMENVATTING

Kunstmatige Intelligentie (KI)-intensieve systemen, met name Computer Vision (CV)-modellen en Deep Reinforcement Learning (DRL)-agenten, worden steeds vaker ingezet in veiligheidskritische toepassingen, zoals autonoom rijden en medische diagnostiek. Hun inzet in deze domeinen vereist grondige testmethoden om betrouwbaarheid en veiligheid te waarborgen, met name omdat er vaak geen duidelijke testorakels beschikbaar zijn, de input gevoelig is voor kleine verstoringen, de testruimte enorm en complex is, en het gedrag dynamisch en stochastisch is — uitdagingen die traditionele testbenaderingen te boven gaan.

In dit proefschrift hanteren wij een teststrategie die puur black-box is, wat beter aansluit bij realistische inzetscenario's waarin de interne werking van modellen niet toegankelijk is. Wij ontwikkelen innovatieve testmethodologieën die bewezen Search-Based Software Testing (SBST)-technieken combineren met benaderingen die gebruikmaken van surrogatemodellen en Grote Taalmodellen (LLMs). Allereerst presenteren wij een puur differentiële-evolutiebenadering voor het genereren van minimale adversariale verstoringen, die wij uitbreiden tot multi-criteria testmethoden die een afweging mogelijk maken tussen de effectiviteit van een adversariale ingrepen en de onzichtbaarheid ervan, om zo een breder scala aan inputkwetsbaarheden bloot te leggen. Bouwend op deze basis introduceren wij een multi-criteria testkader voor Deep Reinforcement Learning dat gelijktijdig optimaliseert voor het detecteren van fouten en de diversiteit van testscenario's, met inzet van lichtgewicht surrogatemodellen. Hierdoor worden ontdekte fouten verspreid over een breder spectrum van agentgedragingen, in plaats van dat slechts specifieke delen van het systeem worden uitgebuit. Vervolgens onderzoeken wij het gebruik van geometriegebaseerde afbeeldingsrepresentaties om de selectie van testgevallen in testsuites voor autonoom rijden te begeleiden, waarbij wij convolutionele neurale netwerken (CNNs) combineren met optimalisatietechnieken voor selectie en prioritering van testgevallen. Ten slotte onderzoeken wij hoe LLMs automatisch softwareontwikkelfouten kunnen oplossen binnen een reproduceerbaar testkader, waardoor het correct testen van KI-intensieve systemen mogelijk wordt zonder tijdrovend handmatig debuggen.

De resultaten van onze experimenten tonen aan dat benaderingen die meerdere multi-criteria optimaliseren, effectiever zijn in het ontdekken van problemen over een breder scala aan scenario's, terwijl zij tevens computationeel efficiënt blijven. Bovendien demonstreren wij dat surrogatemodellen SBST-technieken effectief kunnen verbeteren door het gedrag van het oorspronkelijke systeem onder test te benaderen. Surrogatemodellen bieden een efficiënte fitnessfunctie op basis van hun voorspellingen, die het zoekproces stuurt, aangezien surrogatemodellen de uitvoer van het

model nauwkeurig benaderen zonder dure evaluaties op het oorspronkelijke systeem. Tot slot tonen wij aan dat LLMs het testproces kunnen ondersteunen door middel van natuurlijke taalverwerking om softwareontwikkelingsfouten op te sporen en op te lossen, wat reproduceerbare systeemvalidatie mogelijk maakt.

Kortom, dit proefschrift draagt nieuwe hybride zoekgebaseerde benaderingen bij voor het testen van KI-systemen, waarbij de uitdagingen die inherent zijn aan het testen van CV-modellen en DRL-agenten worden aangepakt. Wij tonen de effectiviteit aan van het combineren van surrogatemodellen en LLMs met bestaande SBST-methodologieën en benadrukken het kritieke belang van diversiteit bij het genereren van uitgebreide testsuites voor robuuste KI-systemen.

ACKNOWLEDGEMENTS

Four years may sound like a long time, but those four years will fly by in the blink of an eye when pursuing a PhD. I have been incredibly fortunate to have had the opportunity to pursue this PhD, and I am profoundly grateful to my advisors, `Annibale Panichella` and `Cynthia Liem`, for taking a chance on an old nerd like me.

My academic journey before this however, was quite turbulant. I was never a stellar student, and often found myself at odds with the traditional education system. I struggled to find my place in the academic world, and it wasn't until I discovered my passion for software testing that I truly found my calling. In 2007, upon completing my bachelor's degree, I was fortunate enough to meet `Carl Thomas`, who hired me as a tester at a small company. This job sparked a love for software testing that has only grown over the years. The challenge of testing software is a puzzle that I have been obsessed with solving. Unlike regular software development, testing requires a different mindset, one that is focused on creating unique and novel solutions that ensure software is reliable and of high quality. This passion for testing helped me wholeheartedly apply myself, leading to a successful career in software testing that spanned nearly 15 years.

My first foray into the world of testing for Machine Learning (ML) was during my time at `IBM` in the Netherlands. Here, I worked on testing IBM Watson Visualizations, a product used to generate visualizations from unlabelled data. This was an interesting experience in which we applied traditional software testing techniques to a product that was powered by ML. It was here that I met some lifelong friends, including `Ruben Kieboom`, `Ramon Pisters`, `Ivo Claessons` and `Hubert Hijkers`. Individuals who I have had the pleasure of working with, and learning from, over the years.

Although not directly related to ML, my time at `IBM` in the USA was also instrumental in shaping my technical skills and mindset. It was here, under the management of `James Wilson` and `Arun Antonio`, that I was able to create elaborate test automation frameworks and build pipelines. It was also here that I enjoyed the mentorship of young and talented individuals, whom have now also moved on to bigger and better things.

After my time in the USA, I felt that my career was reaching a plateau. I wanted to focus more on test automation, particularly with Artificial Intelligence (AI) and ML, but testing in industry is a broken system, as most companies rely on manual validation, rather then taking the time to build proper automation frameworks.

In a random turn of events, my partner Kelley Perry, was in search of a PhD program, and in doing so, led me to find the PhD program at TU Delft. I was fortunate enough to be accepted, and even more fortunate to have Annibale and Cynthia as my advisors. They have been incredibly supportive and have provided me with the guidance and resources I needed to succeed in this program. I am grateful for their mentorship and for the opportunity to work with them.

The PhD journey, while rewarding, is short and not without its challenges. Therefore I am grateful to my partner Kelley for her unwavering support and understanding throughout this process. With lots of amazing vegan food, fresh sourdough bread and a cozy home, she has been with me to commisserate the paper rejections and celebreate the victories. Unfortunately, she was unable to join me at any conferences, due to our furry monsters, Buddha, Udon, and Jellybean, who require constant care and attention. Thank you Kelley for your understanding and support throughout and beyond.

I would also like to thank all my colleagues at TU Delft. Despite working remotely, it was always great to see and chat with all of you while grabbing a coffee in the office. Particularly, I would like to thank Mitchell Olsthoorn and Caro Brandt for their support in the early years, specifically with the Go-No-Go procedure. My early roommate Imara Van Dinten and my later roommates Shujun Huang, Ali Asgari and Cathrine Paulsen. Morover, I would like to thank my colleagues from the Multimedia Computing (MMC) group, Jaehun Kim, Andrew Demetriou, Patrick Altmeyer and Marijn Roelvink, for the fruitful conversations in our weekly meetings and yearly retreats. I would also like to acknowledge Tianqi Zhao for her friendly chats on the 5th floor. Furthermore, I would like to shout out to all the people I have met from SERG and MMC who I have had the pleasure of working with over the years.

One last TU Delft shout out goes to Saskia Peters and Kim Boudewijn for all the help with the annoying admin aspects and Ruud De Jong, who is a super friendly person, and has been an incredible help with all my IT related needs.

Finally, I would like to thank my mother, Helen Bartlett. I cannot imagine it was easy raising my sister Gemma and I as a single parent, but you did it with great love. I am grateful for the first computer you bought for me—a Spectrum ZX-81— and the book on Basic programming that came with it. I vividly remember writing out thousands of lines of code on that machine, and being able to play Horace Goes Skiing as a reward for my efforts. I am grateful for the love and support she has given me throughout my life, and for instilling in me a passion for technology and experimentation.

1

INTRODUCTION

Recent years have witnessed tremendous advances in the field of *Artificial Intelligence* (AI) and *Machine Learning* (ML) [189,211]. These advancements have led to the integration of AI into many aspects of our daily lives; from personalized recommendations on streaming services [114], agents that can help index and summarize emails [152], through to large safety critical systems such as medical diagnostics [201] and autonomous driving [43,218].

Among these advances, *image recognition* and *deep learning*-based vision models have emerged as particularly transformative yet challenging-to-validate technologies. Deep learning has surpassed hand-crafted-feature-based methods in *computer vision* (CV) tasks [174,182], enabling systems to label and classify visual information with unprecedented accuracy. However, these models have revealed a critical vulnerability, in that they remain susceptible to adversarial examples—imperceptibly small perturbations to input images that cause dramatically incorrect predictions [77,200]. This phenomenon is particularly concerning in safety-critical applications like autonomous driving, where vision components guide critical decisions. Despite achieving high accuracy in benign conditions, these models can fail catastrophically on slightly perturbed inputs that humans would classify correctly [209].

Expanding upon vision-based systems, *Deep Reinforcement Learning* (DRL) agents have become increasingly prevalent in autonomous and safety-critical systems. Unlike traditional supervised learning models, DRL agents learn policies through interaction with environments, dynamically adapting to complex, stochastic conditions [145,197]. Self-driving cars epitomize this challenge: they must navigate real-world scenarios with high-dimensional decision spaces, continuously adapting to new road conditions, weather, and traffic patterns. Furthermore, edge-case scenarios are a particular threat, as they may occur rarely in training data, but can trigger fatal failures if they occur in later deployment stages [133].

Both image recognition models and DRL agents share a critical property: they are complex ML systems whose internal decision-making mechanisms remain largely

opaque. This inherent opaqueness makes it increasingly difficult to test and validate these AI systems through traditional software testing approaches [209]. The challenge is further compounded by their integration into larger safety-critical software ecosystems, where the AI component is likely accessed through an interface without access to internal states. However, integration testing offers several well established traditional approaches, such as fuzzing [230,231], combinatorial testing [156], and search-based software testing (SBST) [88].

Consequently, we draw inspiration from SBST, which treats testing as an optimization problem. While SBST has been successfully applied to both white-box testing [84,117,135,165,205,231] and black-box testing [132,155,196,198,236], we focus specifically on the black-box paradigm for AI systems. Black-box testing aligns with the operational reality of deployed AI systems: in practice, these models are often embedded within larger systems, where their internal mechanisms are inaccessible to external entities. SBST addresses black-box testing by using search-based algorithms to efficiently explore program input spaces and automatically generate test cases that reveal failures [88]. The advantage of SBST for black-box scenarios is that we may leverage these input-output observations alone, requiring no knowledge of internal logic or gradient information.

However, the application of SBST to AI systems poses a number of unique challenges [228]. The high-dimensional input spaces of images and the complex, stochastic nature of DRL agents create vast search spaces that can easily overwhelm traditional search algorithms. Notably, the design of a fitness function for identifying failures is complicated by the Oracle Problem [131], as the expected output for a given input may be unknown or difficult to define, particularly in black-box scenarios. Without an oracle, distinguishing between merely unusual behavior and true failures becomes highly challenging for the search process.

Therefore, the goal of this thesis is to explore the application of SBST in the context of AI-intensive systems. More specifically, we seek to understand how we can adapt these techniques to ensure the correct and performative evaluations of ML models, particularly in the domains of CV and DRL.

1.1. Background

In this section, we provide a brief overview of the core concepts and techniques that we applied throughout this thesis. We begin with a discussion of search-based software engineering (SBST)—given its application to all chapters—followed by a brief history of Computer Vision (CV), the predominant ML domain to which we apply a number of research techniques. Next, we explore the realm of Deep Reinforcement Learning (DRL) in self-driving vehicles, a critical area of research in a number of chapters. Finally, we explore the domain of Large Language Models (LLMs), a popular and emerging technology that we apply later in the thesis.

1.1.1. Search-Based Software Testing

Search-Based Software Testing (SBST) is a specialized field within software engineering that treats testing as an optimization problem. Rather than manually designing test cases, SBST leverages various algorithms to automatically explore input spaces and discover test cases that expose failures. The roots of SBST can be traced back to 1975 [142], when backtracking techniques were first applied to automatically generate test cases by identifying impossible program flows and efficiently generating input variables to cover program paths. These pioneering efforts laid the foundation for later work, and by the early 1990s researchers were applying Genetic Algorithms to test data generation [220,225], demonstrating that evolutionary search could effectively guide test generation toward underexplored code paths.

The field matured when the term "Search-Based Software Engineering (SBSE)" was formally introduced [87]—the broader discipline of treating software engineering problems as optimization problems—which naturally encompassed SBST as a subfield. Following this, a comprehensive guidance on applying search-based algorithms to software testing was provided [88], advocating for multi-objective optimization as the preferred approach for balancing competing testing objectives.

At the core of SBST is the definition of a fitness function; a metric that quantifies the quality of candidate solutions. The fitness function guides search algorithms to iteratively improve candidate solutions by exploring the test input space to find optimal or near-optimal test cases. Popular metaheuristic search-based methods employed throughout the chapters of this thesis span from simple (*e.g.,* Hill-Climbing (HC) [111]), through to more complex methods (*e.g.,* Genetic Algorithms (GA) [75]), and Multi-Objective Evolutionary Algorithms (MOEAs) [55,86,106].

Unlike single-objective methods (HC and GA) that seek one optimal solution, MOEAs produce a set of Pareto-optimal solutions, where no solution dominates another without trade-offs. MOEAs form a substantial methodological component of this thesis, enabling us to balance competing priorities such as maximizing failures while maintaining test diversity.

In Figure 1.1, we present an example of a Pareto front, in which we optimize two objectives simultaneously, maximizing both failure and diversity. The points on the Pareto front represent non-dominated solutions [55], where no solution outranks another without accounting for objective trade-offs. In this example, we show the extreme points: Optimal Failure, Optimal Diversity, and Knee Point, along with all non-dominated solutions in between. The optimal points represent the best possible solutions for each objective, while the knee point represents the best trade-off between the two objectives. The knee point is often considered the most desirable solution, as it balances the two objectives without sacrificing too much of either one.

Multi-objective optimization techniques are particularly effective in areas such as CV, where testing must account for competing priorities, such as minimizing adversarial perturbations while modifying the model's prediction to an incorrect class.

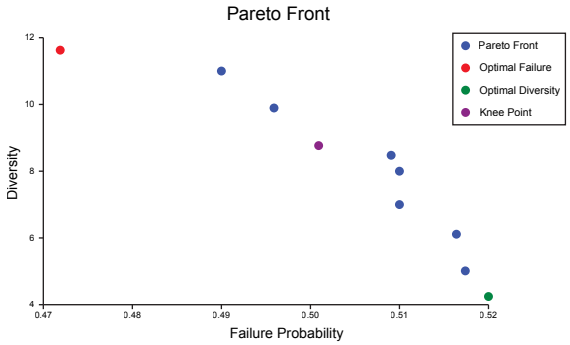


Figure 1.1. Example of Pareto Front optimizing for both failure and diversity.

CV systems—ranging from image classifiers to self-driving vehicles—are highly sensitive to small input changes, making robust testing methodologies essential.

1.1.2. Computer Vision

Computer Vision (CV) is a field of AI that enables machines to label and classify visual information from the world. This area of research presents unique challenges for testing given the sensitivity of CV systems to even minor input variations [77,200], such as misclassified objects in autonomous vehicles or medical imaging [136,141]. In this thesis, CV serves as both a primary focus and a methodological tool, as we generate adversarial examples for image classification models and apply their outputs, such as classification results, as fitness functions for search-based testing techniques. Below, we provide a brief history of the field and its evolution over the years.

The origins of CV can be traced to early research in the 1960s, which introduced foundational concepts for recognizing three-dimensional objects from two-dimensional images and formalized the notion of edge detection in visual data processing [170,171]. These contributions laid the groundwork for subsequent developments in the 1970s, where researchers advanced the field by developing novel approaches to object recognition and image understanding [222]. This period saw the introduction of methods for classifying edges in line drawings based on physical constraints, such as convex, concave, or occluding edges [48,103]. Building on these ideas, further work defined physical labels for edges and used them to infer the three-dimensional structure of objects, inspiring later research in scene understanding and object recognition [215].

With the advent of Deep Learning (DL) techniques in the 2010s, CV experienced a transformative leap; the introduction of convolutional neural networks (CNNs) [127] revolutionized the field, enabling models to learn hierarchical representations

of visual data and achieve unprecedented accuracy on tasks such as image classification, object detection, and semantic segmentation [174,182]. Launched in 2010, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [174] provided a benchmark for evaluating algorithms on object detection and image classification tasks using the ImageNet dataset [58]. The competition not only accelerated innovation, but also demonstrated the superiority of CNN-based models over traditional methods [123,188]. The breakthrough performance of CNNs in ILSVRC spurred further research, leading to architectural refinements such as residual connections, batch normalization, and attention mechanisms.

Today, CV systems powered by CNNs are deployed across diverse sectors, including healthcare, security, and automation [34,140,145]. In healthcare, CNNs analyze medical imaging data for early disease detection, while in autonomous vehicles, they enable real-time object recognition and scene interpretation. Looking ahead, the field continues to evolve with the integration of transformers and other emerging architectures, which are beginning to complement or replace CNNs in specific tasks [172]. There is also a growing emphasis on developing models that are efficient, interpretable, and robust to adversarial inputs, ensuring their reliability in critical applications.

1.1.3. Deep, Deep and Deeper

Deep Reinforcement Learning (DRL) represents the convergence of Reinforcement Learning (RL) and Deep Learning (DL), leveraging the strengths of both fields to enable agents to learn complex behaviors in high-dimensional environments. This section introduces the foundational concepts of DL and RL, before discussing their integration in DRL and its application in this thesis.

1.1.3.1. Deep Learning and Deep Neural Networks

DL [76,178] is a subfield of machine learning that employs Deep Neural Networks (DNNs)—multi-layered architectures capable of learning hierarchical representations of data. DNNs have revolutionized fields such as image recognition [92,183], natural language processing [96], and game playing [36], by automating feature extraction and modeling complex patterns. However, their opacity and internal complexity introduce significant challenges for black-box testing, particularly in safety-critical applications [85,209]. Due to the high dimensionality of their input spaces and the non-linear interactions between layers, traditional testing techniques struggle to effectively explore the decision boundaries of DNNs or identify failure modes.

1.1.3.2. Reinforcement Learning

Reinforcement Learning (RL) is a form of ML where an agent learns to make decisions by interacting with an environment [197]. We model basic RL as a Markov Decision Process, which consists of a set of states, actions, and rewards. The agent

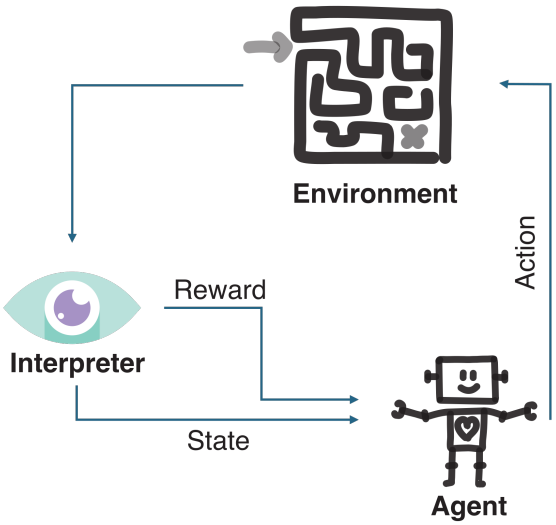


Figure 1.2. Example reinforcement learning.

learns to take actions that maximize the cumulative reward over time by exploring the environment and receiving feedback in the form of rewards or penalties. In Figure 1.2, we present a visual representation of the RL process, where an agent interacts with an environment by taking actions and receiving rewards based on those actions.

1.1.3.3. Deep Reinforcement Learning

DRL addresses the limitations of traditional RL by combining it with the representational power of DNNs. This integration enables agents to handle high-dimensional input (e.g., raw pixels from cameras) and learn policies directly from sensory data. DRL has achieved remarkable success in domains such as robotics [195,204] and autonomous driving [118,202].

We have observed DRL increasingly applied in the domain of self-driving cars (SDCs) [145], where agents are trained to navigate complex, simulated environments by learning policies that maximize rewards associated with safe and efficient driving behaviors [208,219]. A key application within this domain is the development of lane-keeping agents, which are tasked with maintaining the vehicle within lane boundaries using real-time input from onboard sensors, such as cameras. The inherent stochasticity of DRL agents, combined with the dynamic and often unpredictable nature of driving environments, presents significant challenges for ensuring the robustness and reliability of such systems [44,166]. These challenges are particularly pronounced in

safety-critical scenarios, where the ability to generalize learned behaviors and adapt to edge cases is essential for operational success.

In Figure 1.3 [30], we display two scenarios of a DRL agent performing lane-keeping in the BeamNG simulation environment [18]. We train the DRL agent to keep the vehicle within the lane boundaries, and we assign higher rewards for staying near the lane center and penalties for leaving the lane. In the first image (a), the vehicle sits comfortably within the lane, while in the second image (b), the vehicle has drifted over the lane and is now unsafely on the shoulder of the road.

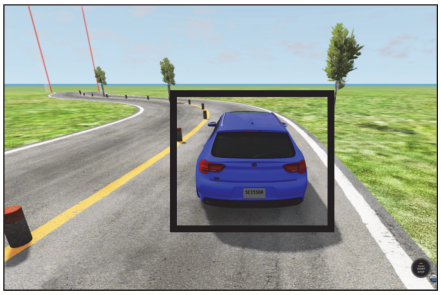
1.1.4. Simulation Environments

Simulation environments are indispensable in the development and testing of AI systems, particularly for DRL agents operating in complex, dynamic domains like autonomous driving [113,164]. These environments provide a safe, controlled, and cost-effective platform, for the training and evaluation of ML agents. This is especially important in safety-critical domains, such as autonomous driving, where real-world testing can be costly, dangerous, and considerably time-consuming. Simulation environments, such as BeamNG.tech [18] and CARLA [65], offer researchers and practitioners the ability to train, test, and validate our agents in a complex virtual setting. These environments provide realistic physics, detailed graphics, sensor integrations, including LiDAR, GPS and cameras, to support the deployment of multi-agent systems. Due to their flexibility and realism, simulation environments have become an essential tool in our development of SDCs [8,93,112,238].

1.1.5. Large Language Models

Large Language Models (LLMs) are transformer-based deep learning models trained on vast corpora, enabling them to excel at code generation [10,42,105,130] and interpret unstructured signals such as error messages and build logs. Their transformer architectures and attention mechanisms enable them to capture long-range dependencies, making them particularly effective at correlating error messages with code context, aggregating multiple error signals, and reasoning about constrained optimization problems such as code generation [37,221]. While full-scale LLMs demand substantial computational resources, model compression techniques [95,108,216] enable efficient deployment on modest hardware, making them accessible for offline operation and reproducible research without requiring expensive cloud infrastructure or external API calls.

These capabilities make LLMs valuable for tasks like error analysis and code repair, particularly in resource-constrained research environments where reproducibility is essential. By leveraging open-source compressed models that can be version-controlled and frozen at specific checkpoints, researchers can maintain identical model weights and code versions across time and institutions, addressing fundamental challenges in scientific computing [95,108].



(a) Lane keeping (success)



(b) Lane keeping (failure)

Figure 1.3. Example DRL agent lane-keeping scenarios in a simulated environment. In (a) the agent is successfully following the lane-keeping scenario, In (b) the vehicle has deviated from the lane, causing a failure. Source: Birchler *et al.* 2024.

1.2. Challenges

The evolution of this thesis reflects a journey through several critical challenges in the testing and validation of AI-intensive systems. Our initial experimentation with adversarial inputs revealed not only the vulnerabilities of image recognition models, but also the broader challenges of testing AI systems in general.

1.2.1. The Black Box Nature of AI-Intensive Systems

The opaque nature of machine learning models, particularly deep neural networks (DNNs) and reinforcement learning (RL) agents, poses a fundamental challenge for testing and validation. Unlike traditional software systems, where the logic is explicitly encoded in code, AI systems encode their behavior implicitly through learned parameters rather than explicit logic. This creates a paradigm shift in testing: for many novel inputs, there is no predetermined "correct" answer, and the model's output becomes the prediction we must evaluate. This absence of a defined ground truth exacerbates the inherent Oracle Problem [131] and creates a paradox central to AI testing; we seek to find inputs that expose failures, yet we have no objective standard against which to judge whether the system has failed.

When testing AI systems, practitioners must choose between numerous testing approaches. Black-box techniques [155,236] are commonly chosen for this task, as they more accurately reflect real-world deployments where systems are accessed solely via inputs and outputs, removing access to internal states, intermediate representations, or gradients. Without this introspection, correctness is judged solely from external behavior, making it difficult to interpret the model's output as a true failure or an acceptable but unexpected outcome.

In contrast, white-box testing techniques [135,205], which do have access to internal states, have been proposed for testing AI systems. However, they have been shown to perform inadequately in practice [5]. Furthermore, white-box testing is often infeasible in real-world scenarios, where models are likely proprietary [3].

Consequently, our focus on black-box testing introduces an opacity challenge that directly impacts our use of search-based software testing (SBST) techniques. SBST relies on a fitness function to guide the search process toward failure-inducing inputs, but the lack of a clear oracle makes defining this fitness function a significant challenge.

1.2.2. Simulation environments and validation runtime

Effective validation of DRL agents requires executing them repeatedly within simulation environments, which introduces significant runtime constraints that limit the scalability and comprehensiveness of testing efforts. While simulation environments are essential for safe development, as they reduce the necessity of real-world testing, they present their own computational challenges when used extensively for validation [51,78].

Modern simulation environments supporting DRL agent testing span diverse domains and fidelities. At one end, lightweight environments like OpenAI Gym [36] provide fast, basic simulations suitable for algorithm development. At the other end, high-fidelity simulators such as BeamNG [18] and CARLA [65] model complex physics, realistic sensor behaviors (LiDAR, cameras, GPS), graphics rendering, and intricate environmental interactions. These varied environments serve different purposes, from rapid prototyping, to realistic validation, but all impose runtime costs that escalate significantly during validation.

The runtime burden becomes compounded when applying search-based testing techniques, as each candidate scenario involves executing the DRL agent within the simulated environment to validate the outcome, requiring seconds to minutes of simulation time per scenario. This process, which must be repeated for hundreds or thousands of candidate scenarios to discover failures, can quickly become prohibitive. For researchers and practitioners with limited access to high-performance computing resources, this runtime barrier significantly constrains the breadth of validation efforts and the diversity of scenarios that can be thoroughly tested. The challenge is particularly prominent when attempting to validate policies across multiple environments or to perform real-time or near-real-time testing during deployment preparation.

1.2.3. High-dimensional search spaces and SBST

When applying SBST techniques to AI-intensive systems, practitioners face challenges related to the dimensionality and structure of the search spaces involved.

SBST's core strength lies in its ability to efficiently explore input spaces using metaheuristic algorithms [88]; however, this strength can become a liability when confronted with high-dimensional spaces characteristic of modern AI systems [151].

Consider, for example, an autonomous driving agent: the agent controls a vehicle through continuous action spaces encompassing steering angles, acceleration rates, and braking forces. Unlike discrete action spaces (e.g., "turn left," "turn right"), continuous action spaces are infinite, creating a search landscape of unbounded dimensionality. Even when reducing the action space to a discretized version, say 100 distinct steering angles and 50 acceleration levels, the resulting search space becomes enormous [125].

These high-dimensional landscapes present a fundamental challenge to SBST methodologies. Traditional single-point search algorithms, such as Hill-Climbing, risk converging prematurely to local optima, a problem further exacerbated by the complex, non-convex structure of the fitness landscapes in AI systems. Population-based approaches, such as Genetic Algorithms (GAs) and Multi-Objective Evolutionary Algorithms (MOEAs), offer a partial remedy through population diversity [187]; by maintaining a diverse population of candidate solutions, these algorithms can explore multiple regions of the search space simultaneously and collectively avoid narrow local optima.

However, maintaining sufficient population diversity in high-dimensional spaces is itself a non-trivial challenge. When the search space grows to thousands or millions of dimensions, the volume of unexplored regions far exceeds what a finite population can cover. Even with mutation and recombination operators designed specifically to promote exploration, genetic drift [210] becomes more pronounced as the population's solutions cluster in subregions of the space, reducing the ability to discover novel and diverse failure cases [116]. Furthermore, the effectiveness of mutation as a diversity-maintenance mechanism diminishes in high dimensions: random perturbations in 150,000-dimensional space are likely to move solutions only marginally, potentially missing important regions of the landscape entirely.

1.2.4. Test case diversity

Diversity in generated test cases is crucial for effective testing [133]. Non-diverse test suites risk clustering around similar scenarios, leaving edge cases untested and potentially missing critical failures. This is particularly important in safety-critical domains like autonomous driving or medical diagnostics, where undetected failures can have catastrophic consequences [72,80,85,124,214].

The diversity challenge manifests in multiple ways. First, search-based testing algorithms, when optimizing solely for failure detection, naturally converge toward exploiting specific vulnerabilities rather than uncovering a spectrum of weaknesses. Second, when working with high-dimensional input spaces, such as those in DRL agents, the vast majority of the space remains unexplored, and simple random

sampling cannot guarantee adequate coverage of behavioral variations. Third, different failure modes often require fundamentally different test cases: a CNN classifier may fail due to imperceptible pixel perturbations, texture manipulations, geometric transformations, or combinations thereof, yet a narrowly-focused search may discover only one class of failure [200].

For example, a DRL agent tested solely on straight roads would likely fail when tested on curved roads [119]. Similarly, an image classifier trained on well-centered objects may fail on off-center or partially occluded inputs [200]. Approaches that optimize for failure alone will likely generate test cases that exploit a specific weakness in the model, rather than broader weaknesses. This tendency is particularly common when employing search-based techniques, which have a tendency to converge on local optima, clustering test cases around easily discoverable vulnerabilities and neglecting less obvious failures that may be more critical [88]. Therefore, we must maintain diversity in our generated test cases to ensure that a comprehensive range of scenarios and failure modalities are covered.

1.3. Research Goals & Questions

This thesis seeks to address four interrelated research questions (RQs), each targeting multiple challenges, while advancing the overarching goal: adapting and enhancing traditional software testing techniques for AI-intensive systems. Below, we outline these RQs, their motivations, and their connections to our methodological approach (Section 1.4), and empirical chapters.

We begin with the fundamental challenge of generating effective test cases for AI systems. This involves generating adversarial examples that can expose vulnerabilities in models while maintaining realism to avoid detection. Therefore, we seek to answer the following research question:

Thesis Research Question 1

To what extent can search-based software techniques be applied to the generation of adversarial examples with minimal perturbation?

Minimizing perturbations is important, as it provides realism to the generated adversarial examples, reducing the possibility of triggering adversarial detection by the model. However, if the generated adversarial examples are not diverse enough, then we may end up generating examples that exploit only a specific weakness in the model rather than a broader weakness. For example, testing a `computer vision` (CV) model with only texture-based attacks, may lead to a model weak against geometric attacks. Therefore, we seek to answer the following research question:

Thesis Research Question 2

How does co-optimizing for diversity improve the effectiveness and generality of adversarial examples?

We must selectively minimize test suites while preserving effectiveness—a process known as *Test Case Selection* (TCS) and *Test Case Prioritization* (TCP). The objective is to select and execute only the most failure-prone or diverse test cases, avoiding redundant evaluations without sacrificing coverage quality. However, traditional TCS/TCP methods, such as genetic algorithms, rely on handcrafted fitness functions (*e.g.,* code coverage or failure history), which often fail to capture complex, non-linear behaviors of AI systems. This limitation is compounded by the challenge of black-box testing (Section 1.2.1), where unknown expected outputs make it difficult to design effective fitness metrics.

Recently the use of surrogate models has emerged as a promising approach to simulate the behavior of the larger target system. These surrogate models are lightweight approximations that can be more efficiently evaluated, enabling faster iterations in the search process. By leveraging surrogate models, we can potentially guide the search process more effectively, even in the absence of detailed internal knowledge of the target system. This approach raises an important question about the impact of surrogate models on the quality of test case selection:

Thesis Research Question 3

How does the use of surrogate models impact the quality of test cases selected through search-based software testing techniques?

Beyond generating and selecting test cases, ensuring reproducibility of both our testing methodologies and the models themselves is foundational to scientific validity. Reproducibility challenges extend beyond individual model testing to the broader software ecosystem supporting AI systems. The rapid advancement of dependency software and frameworks creates fragility in research projects and testing infrastructures. When different modules require incompatible versions of the same dependency, either directly or transitively through nested dependencies, runtime failures inevitably occur [226]; fundamentally hindering the reproducibility of both our testing approaches and the models we test. Existing approaches to dependency management, such as knowledge-graph-based techniques, require perpetual maintenance and struggle to generalize across the evolving software landscape [45,227]. To establish a robust foundation for reproducible AI system testing and validation, we ask:

Thesis Research Question 4

How can machine learning models help enable reproducible testing of AI-intensive systems?

1.4. Research Methodology

The research methodology set out in this thesis is heavily influenced by the practical challenges and constraints inherent in testing AI-intensive systems, such as black-box testing, high-dimensional search spaces, and computational cost of simulation environments. Unlike traditional research approaches that may prioritize theoretical exploration in isolation, our methodology is rooted in the real-world demands of software engineering and testing, particularly in the context of *computer vision* (CV) and *deep reinforcement learning* (DRL). This dual focus—on both academic rigor and practical relevance—ensures that the artifacts we develop are not only theoretically sound but also deployable in operational environments.

Our approach emphasizes the creation and rigorous evaluation of novel artifacts, methods, algorithms, and tools, that directly address the identified challenges in AI-intensive system testing. We combine empirical software engineering techniques with iterative development practices, enabling us to systematically explore research questions while maintaining flexibility to adapt as new insights emerge. This problem-driven methodology prioritizes practical applicability and reproducibility, ensuring that our solutions are grounded in evidence and accessible to both researchers and practitioners.

1.4.1. Research Principles

We are guided by the following principles throughout every research chapter of this thesis:

Problem-centric and exploratory

Each chapter in this thesis begins by identifying a concrete problem that merits investigation. This problem-centric approach ensures that our research efforts remain grounded in meaningful challenges and produce actionable solutions. In practice, we conduct small-scale preliminary experiments to explore the problem space, experiment with available technologies, and identify key focus areas that warrant deeper investigation. Through this exploratory process, we uncover specific research questions that guide subsequent empirical work. For example, Chapter 2 began with the observation that existing adversarial attack methods often generate unrealistic perturbations, leading us to explore methods that flip a models predictions with minimal changes.

ITERATIVE EXPLORATION AND ARTIFACT DEVELOPMENT

The process of advancing our research is inherently iterative, with each cycle generating artifacts designed to address our research questions. Rather than attempting to solve problems perfectly on the first attempt, we embrace an iterative refinement approach where preliminary artifacts are evaluated, insights are extracted, and subsequent iterations incorporate these learnings. This methodology enables continuous improvement through empirical feedback. In Chapter 5 for instance, we began with simple road plot representations for CNN-based test case selection, then iteratively explored geometric heatmap and point-based encodings, progressively refining our understanding of which image representations best capture failure-indicative features. The evaluation-driven iteration process ensures that each artifact iteration moves us closer to effectively answering our research questions while maintaining practical applicability.

RIGOROUS EMPIRICAL EVALUATION

Validating the effectiveness of our artifacts requires systematic empirical investigation using established experimental methodologies. Throughout this thesis, we implement various evaluation approaches tailored to each research question, including baseline comparisons against state-of-the-art techniques, statistical significance testing using non-parametric methods (Mann-Whitney U, Wilcoxon signed-rank tests), effect size quantification (Vargha-Delaney $\widehat{A}_{12}$), and ablation studies to isolate component contributions. This rigorous approach ensures that research questions are answered with evidence rather than intuition, and that the results generated by our artifacts meet clearly defined requirements. Each chapter includes comprehensive empirical evaluation.

OPEN AND REPRODUCIBLE RESEARCH

A foundational principle underlying all our work is the commitment to open science and reproducibility. We prioritize implementing only open-source software, using publicly available datasets, and employing open-source ML models wherever possible. All artifacts developed throughout this thesis are released under permissive open licenses, accompanying our published papers with full replication packages including source code, datasets, experimental configurations, and detailed documentation. This commitment to openness serves multiple purposes: it enables independent verification of our results, lowers barriers to adoption by researchers and practitioners, and facilitates scientific progress by allowing others to extend, adapt, and build upon our work. As discussed in detail in Section 1.7, this approach reflects our belief that reproducibility and transparency are not optional add-ons, but fundamental pillars of rigorous research.

Table 1.1. Thesis research questions and related chapters.

RQ	Research Question	Chapters
RQ1	To what extent can search-based software techniques be applied to the generation of adversarial examples with minimal perturbation?	2
RQ2	How does co-optimizing for diversity improve the effectiveness and generality of adversarial examples?	3,4
RQ3	How does the use of surrogate models impact the quality of test cases selected through search-based software testing techniques?	4
RQ4	How can machine learning models help enable reproducible testing of AI-intensive systems?	5

1.5. Research Outline

In this section, we provide a brief overview of the chapters included in this thesis, in which each chapter seeks to address one or more of the research questions outlined in Section 1.3. Table 1.1 illustrates the connections between the various research questions and our chapters.

Chapter 2: Adversarial examples—inputs that cause deep neural networks to make incorrect predictions through imperceptibly-small modifications—represent a critical vulnerability in image classification systems, particularly in safety-critical applications such as autonomous vehicles [77,200]. We can classify existing adversarial attack methods into white-box approaches that require access to model internals and black-box methods that require only input-output queries [132]. The existing state-of-the-art black-box approach, `BMI-FGSM` (Black-box Momentum Iterative Fast Gradient Sign Method), combines *Differential Evolution* (DE) with gradient approximation but requires thousands of model queries and complex parameter tuning [132]. This chapter proposes a pure black-box attack strategy based exclusively on DE, eliminating the need for gradient approximation and significantly reducing the number of required iterations. We introduce three specialized mutation operators (add, delete, and change), designed specifically for image perturbation, and extend our approach to a multi-objective formulation that jointly minimizes the number of pixel modifications while flipping the model's prediction. To evaluate the effectiveness of our approach, we implement multiple algorithm variants including single-objective DE, NSGA-II [57], NSGA-III [56], and AGE-MOEA [161], and introduce a novel post-processing technique that gradually reverts adversarial changes back toward original pixel values while preserving the attack's effectiveness. We evaluate our approach on five state-of-the-art deep neural network models (VGG16, VGG19, ResNet50, ResNet101, ResNet152) using 150 images sampled from the ImageNet ILSVRC2012 validation dataset. Our results demonstrate that our evolutionary approach generates adversarial examples with significantly fewer pixel perturbations compared to `BMI-FGSM` while completing the search process substantially

faster. The multi-objective formulation proves particularly effective, enabling us to explore trade-offs between attack strength and perturbation minimization, and giving practitioners fine-grained control over the adversarial example generation process.

Chapter 3: While search-based techniques have shown promise in generating failure-inducing test scenarios for Deep Reinforcement Learning (DRL) agents [209], existing approaches focus primarily on maximizing failure detection without considering the diversity of discovered failures. The INDAGO framework [24], for example, uses lightweight surrogate models trained on DRL training logs to predict failure likelihood, but overlooks the importance of generating diverse test scenarios; a key consideration in testing, as diversity promotes broader coverage of the agent's operational space and improves the chances of uncovering unique faults. Chapter 3 addresses this limitation by proposing INDAGO-Nexus [15], a multi-objective search framework that jointly optimizes for failure probability and scenario diversity. INDAGO-Nexus employs Multi-Objective Evolutionary Algorithms (MOEAs), specifically NSGA-II and AGE-MOEA, to explore trade-offs between the two objectives and produce a set of Pareto-optimal solutions. The framework incorporates two diversity metrics—Euclidean distance and PCA-based clustering [64]—and evaluates two Pareto-front selection strategies including the knee-point method [234], which identifies the solution providing the best trade-off between objectives. We benchmarked INDAGO-Nexus on three DRL agents representing different application domains: self-driving cars, humanoid walkers, and parking agents. Our empirical evaluation demonstrates that INDAGO-Nexus discovers significantly more unique failures than INDAGO, achieving up to 83% and 40% more unique failures in the SDC and Parking scenarios respectively, while also reducing time-to-failure by up to 67%. The multi-objective approach, combined with PCA-based diversity metrics and knee-point selection, consistently produces the best results across most configurations, providing a more comprehensive and efficient testing methodology for DRL agents in safety-critical domains.

Chapter 4: Test case selection for self-driving cars is critical for efficient simulation-based testing, as executing all generated scenarios is prohibitively expensive [27]. While conventional ML approaches leverage sequential models and engineered geometric features for failure prediction [9,82], they do not explicitly capture the spatial structure inherent in road layouts. This chapter investigates geometry-based image representations as an alternative paradigm for SDC test case selection; motivated by the observation that CNN architectures can effectively extract spatial features from geometrically-encoded images [53]. We conduct a large-scale empirical analysis of approximately 10,000 road scenarios, revealing that abrupt geometric changes, such as jerk, chord deviation, and path angle variation, are more predictive of failures than absolute curvature values alone [25]. Guided by these findings, we curate three distinct image datasets (simple road plot, geometric heatmap, and geometric points) and optimize six CNN architectures, including fine-tuned VGG16/VGG19 models and custom architectures adapted from the SnakeCLEF competition submissions [167,168]. We implement an adaptive diversity-aware selection algorithm that bal-

ances predicted failure likelihood with coverage of distinct test scenarios, addressing the challenge of test suite diversity in SDC testing. We evaluate our approach on a newly created dataset of approximately 16,000 test cases and compare it against state-of-the-art SDC test selection techniques, including ITS4SDC [82], RoadGeom [9], and DRVN [17]. Our results demonstrate that geometry-based CNN models achieve competitive fault detection efficiency on key effectiveness metrics, with our EfficientNet-based model obtaining superior time-to-fault ratios, while custom geometric CNNs consistently obtain higher test suite diversity. These findings indicate that image-based representations of road geometry, when combined with structure-aware models and adaptive diversity handling, effectively support test case selection for self-driving cars in simulation-based testing environments.

Chapter 5: Dependency conflicts represent a critical barrier to the reproducibility and maintainability of Python research projects. When different modules require incompatible versions of the same dependency, either directly or transitively through nested dependencies, runtime failures inevitably occur [226]. This challenge is particularly acute in ML and scientific computing, where libraries frequently depend on hardware-specific versions or require tightly coupled version combinations [102]. Existing knowledge-graph-based approaches such as PyEGO and ReadPyE [45,227] construct comprehensive databases of module relationships but suffer from a fundamental limitation: they require frequent and labor-intensive updates to remain effective as the Python ecosystem rapidly evolves [150]. This chapter addresses the problem by introducing PLLM, a novel LLM-based approach that leverages Retrieval-Augmented Generation (RAG) to dynamically resolve dependency conflicts through iterative repair, driven by concrete build error feedback. Our key insight is that Python error messages encode rich semantic information about missing and conflicting dependencies, enabling LLMs to infer working dependency configurations without relying on static knowledge graphs. We evaluate PLLM on the Gistable benchmark, a curated collection of 2,891 real-world Python programs with known dependency issues [98], comparing it against state-of-the-art knowledge graph baselines. Our results demonstrate that PLLM outperforms PyEGO by 21.58% and ReadPyE by 15.97% in the number of successfully resolved dependencies, while being particularly effective for complex dependencies commonly encountered in ML workflows such as TensorFlow, NumPy, and SciPy. This LLM-driven approach eliminates the need for pre-built infrastructure and significantly improves generalization across diverse dependency error types.

Chapter 6: In this final chapter, we revisit the thesis research questions and summarize the key findings, formulating conclusions based on the results obtained throughout the thesis. Furthermore, we discuss the limitations of the work and provide suggestions for future research directions in the field of testing AI-intensive systems.

Table 1.2. Thesis chapter artifact publication details.

Chapter	Host	Replication Package DOI
2	Zenodo	https://doi.org/10.5281/zenodo.10250866
3	Figshare	https://doi.org/10.6084/m9.figshare.29204687
4	Figshare	https://doi.org/10.6084/m9.figshare.30906746
5	Figshare	https://doi.org/10.6084/m9.figshare.29204693

1.6. Origins of Chapters

The chapters in this thesis draw on papers we authored during our Ph.D. journey, and those papers form the foundation of this work. Peer reviewers reviewed, and venues published, the included papers across several conferences and journals, with Chapter 4 as the only exception; it is currently under review with the Journal of Systems and Software (JSS). Readers can therefore approach each paper in this portfolio-style thesis as a standalone work with dedicated background, motivation, related work, and conclusion sections. We adjusted each paper to better fit the style and format of this thesis, including table layouts and image formatting. Despite these adjustments, we have not altered the main content of each paper. We are the lead authors of all included papers.

Table 1.2 provides an overview of the digital object identifiers, and replication packages for each thesis chapter.

- **Chapter 2** appears in the paper "Multi-objective differential evolution in the generation of adversarial examples", published in Volume 238 of the Science of Computer Programming journal [12].
- **Chapter 3** appears in the paper "The Pursuit of Diversity: Multi-Objective Testing of Deep Reinforcement Learning Agents", presented at the 2025 SSBSE Workshop, co-located with the 40th IEEE/ACM International Conference on Automated Software Engineering (ASE) [15].
- **Chapter 4** is currently under review with the Information and Software Technology Journal (IST) as the paper "Snakes On a 2-D Plane: Test Case Selection for Autonomous Driving Simulation Using Image-Based Representations".
- **Chapter 5** appears in the paper "The Last Dependency Crusade: Solving Python Dependency Conflicts with LLMs", presented at the 2025 Inaugural AgenticSE Workshop, co-located with the 40th IEEE/ACM International Conference on Automated Software Engineering (ASE) [14].

1.7. Open Science

A foundational barrier to testing software systems, including ML systems, is reproducibility. The rapid advancement of software, often renders older code incompatible

with newer environments [98,226]. This systemic issue persists, as researchers often lack foundational software engineering skills for building and maintaining robust software. Under these circumstances, testing ML systems becomes more difficult, as minor library differences can significantly change model performance and behavior, making cross-study comparisons difficult.

In this thesis, we commit to the principles of open science, that emphasize transparency, accessibility, and reproducibility in research [223]. By making all our research artifacts, datasets, and software tools publicly available, we enable independent verification of our results, lower barriers to adoption, and facilitate scientific progress in the broader community. Open science practices increase research impact and accelerate innovation by allowing other researchers and practitioners to build upon, extend, and adapt our work [68].

1.7.1. Open Data

Chapter 4 contributes a newly curated dataset of approximately 16,000 generated road scenarios for DRL agent testing. This comprehensive dataset enables reproducible empirical validation of test case selection techniques and is publicly available to support future research and educational endeavors in autonomous driving and simulation-based testing.

1.7.2. Open Software

We release all software developed in this thesis as open-source tools under permissive licenses. We developed the PLLM tool (Chapter 5), which automates dependency resolution for Python projects using LLMs. To advance adoption of open-source agentic engineering, we deployed PLLM as the baseline for the FSE-AIware'26 competition "Agentic-Based Python Dependency Resolution Competition" [13]. This inaugural competition attracted two teams from academia, who submitted approaches to resolve real-world dependency conflicts. We evaluated these approaches using a newly created dataset of Python projects, which we also released as part of the competition proceedings.

We provide a replication package for each chapter in this thesis (Table 1.2), which includes full source code, datasets, experimental results, and documentation. These packages enable independent verification of our results and facilitate further research by allowing others to build upon our work.

1.8. Other Contributions

In addition to the publications included in this thesis, we present several additional co-authored papers, which are described below:

- In the ICML'24 position paper "Position: stop making unscientific AGI performance claims" [6], we argue that observing meaningful patterns in the latent spaces of ML models cannot be taken as evidence of Artificial General Intelligence (AGI). We empirically demonstrate that a range of models, including random projections, matrix decompositions, deep autoencoders, and transformers, can distill information that correlates with external variables without possessing AGI. We also review social science literature showing that humans are prone to seek patterns and anthropomorphize, leading to misinterpretations of model capabilities. We call for caution in interpreting and communicating AI research outcomes to avoid unscientific claims about AGI.
- In the SSBSE'24 workshop NIER Paper "Danger is My Middle Lane: Simulations from Real-World Dangerous Roads" [16], we propose a novel approach, using real-world sourced 'dangerous' roads as a seeding point for the generation of test scenarios for self-driving cars. In this preliminary study, we evolve scenarios using a 1+1 Non-Dominated Evolutionary Algorithm, to generate roads that cause Out-Of-Bounds (OOB) failures in a DRL lane-keeping agent. Comparing against a random baseline, we found that our approach created higher instances of OOB failures.
- In the IEEE IV'26 conference paper "Efficient Exploration of Autonomous Driving System Safety Boundaries" [138], we introduce a boundary exploration approach inspired by contour tracing to efficiently reconstruct the collision boundary in two-dimensional scenario spaces. Our approach operates in three phases: initialization, boundary discovery, and boundary tracing using Moore-Neighbor Tracing. We evaluate four variants over 35 search spaces using Autoware as an Autonomous Driving System (ADS), and demonstrate that the approach identifies the safety boundary by sampling less than 11% of the search space, yielding a nearly tenfold increase in efficiency with no loss of boundary fidelity.

I

RESEARCH CHAPTERS

2

MULTI-OBJECTIVE DIFFERENTIAL EVOLUTION IN THE GENERATION OF ADVERSARIAL EXAMPLES

Adversarial examples remain a critical concern for the robustness of deep learning models, showcasing vulnerabilities to subtle input manipulations. While earlier research focused on generating such examples using white-box strategies, later research focused on gradient-based black-box strategies, as models' internals often are not accessible to external attackers. This paper extends our prior work by exploring a gradient-free search-based algorithm for adversarial example generation, with particular emphasis on differential evolution (DE). Building on top of the classic DE operators, we propose five variants of gradient-free algorithms: a single-objective approach (GA$_{DE}$), two multi-objective variations (NSGA-II$_{DE}$ and MOEA/D$_{DE}$), and two many-objective strategies (NSGA-III$_{DE}$ and AGE-MOEA$_{DE}$). Our study on five canonical image classification models shows that whilst GA$_{DE}$ variant remains the fastest approach, NSGA-II$_{DE}$ consistently produces more minimal adversarial attacks (i.e., with fewer image perturbations). Moreover, we found that applying a post-process minimization to our adversarial images, would further reduce the number of changes and overall delta variation (image noise).

This chapter was published in The Elsevier Journal: Science of Computer Programming by Antony Bartlett, Cynthia C. S. Liem and Annibale Panichella [12]. See Chapter 1.6 for additional publication details.

2.1. Introduction

Modern applications increasingly rely on machine learning models for automated decision-making based on observed data patterns.

2

Despite early challenges in processing high-dimensional but semantically low-level multimedia data (such as raw pixel values in images) [185], the computational surge and increasing popularity of deep learning has enabled the direct learning of associations between input data and output labels. Ever since so-called deep convolutional neural networks outperformed hand-crafted methods in the 2012 ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [174], deep learning models have become mainstream in computer vision, but also in many other applied machine learning domains, such as natural language processing and music information retrieval.

Deep learning models have been lauded for yielding high-accuracy predictions and thus, have become attractive candidates for integration in real-life systems that may be safety-critical (*e.g.,* vision components in self-driving cars). At the same time, they have been criticized for making intolerable and sometimes incomprehensible prediction errors, jeopardizing safety. As has been shown in the machine learning world, they are *e.g.,* inherently vulnerable to so-called adversarial attacks, in which perceptually small changes to input data can cause very different, erroneous model predictions [77,200].

Adversarial examples have been extensively investigated in the literature, where the idea is to introduce subtle changes in the data (*e.g.,* changing the pixels in a target image) that do not change the ground truth but make a Deep Learning model predict incorrect output. Existing approaches to adversarial example generation can be classified into white-box and black-box methods. White-box approaches [84,117,135,165,205,231] require access to the model under test (i.e. the model architecture, neuron weight values, and gradients). Black-box strategies [132,155,196,198,236], instead, only require access to model inputs and outputs. These approaches are considered more realistic as it reflects what external attackers can obtain [132], *e.g.,* in the case of remote API access.

In this domain, Lin et al. [132] introduced the *Black-box Momentum Iterative Fast Gradient Sign Method* (`BMI-FGSM`) to approximate the gradient, based on a few data points. These points are obtained by mutating existing images (seeds) with *evolutionary algorithm* (EA), and *differential evolution* (DE) in particular. Their results showed that `BMI-FGSM` outperforms white-box and black-box approaches previously proposed in the literature.

In spite of these undisputed results, we observe that `BMI-FGSM` requires thousands of iterations to successfully generate adversarial attacks, despite the usage of gradient-based methods. In our previous work [11], we compared the state-of-the-art `BMI-FGSM` to our single and multi-objective black-box attack methods, across five image recognition models. This experiment was run across a small set of images (50), to which we discovered the following. Our single-objective approach performed

well in quickly creating adversarials, whilst the multi-objective approach would take slightly longer but create adversarial examples with fewer perturbations.

Through this comparative analysis, we established that employing pure black-box attacks using differential evolution was considerably faster than previous adversarial example generation. Further to this, we were able to generate adversarial examples with significantly fewer perturbations (pixel changes).

Therefore, this paper maintains its focus on black-box strategies targeting deep neural network (DNN) models for image recognition. We investigate the usage of DE alone—i.e., without employing gradient-based methods—as the core technique to generate adversarial attacks in a black-box fashion. In particular, we utilize five customized operators for DE that introduce perturbations in the target images. Notably, we focus on DE as meta-heuristics due to their reported superiority over gradient-based approaches, as observed in recent studies [176].

This work extends the previous by adding a second multi-objective variant *MOEA/D*[1] [232] (`MOEA/D`$_{\text{DE}}$), along with two many-objective variants, *NSGA-III*[2] [56] (`NSGA-III`$_{\text{DE}}$) and *AGE-MOEA*[3] [161] (`AGE-MOEA`$_{\text{DE}}$). These additions, particularly the many-objective variants utilized as part of our multi-objective solution, expand the scope of our investigation.

Additionally, we expand our tests, sampling an extra 100 images from the 2012 ImageNet validation dataset. This culminates in a total of 150 images, with that being the original 50 images and the new 100. We have continued sampling from this particular dataset, as it is the validation data released in conjunction with the data, on which the models under test have been trained.

Subsequently, our exploration delves into the application of post-processing techniques on generated adversarial examples. We employed a randomized minimization algorithm designed to gradually revert the changes applied to the original images while preserving the adversarial capabilities in fooling the targeted DNNs. The goal of this algorithm is to generate a set of less noticeable changes, whilst maintaining a flipped prediction.

We thus focus on the following two research questions:

RQ1 : *How do various black-box evolutionary algorithms perform in generating adversarial examples?*

RQ2 : *How effective is post-processing at improving the quality of our adversarial examples, by minimizing pixel color delta variation?*

[1] https://pymoo.org/algorithms/moo/moead.html
[2] https://pymoo.org/algorithms/moo/nsga3.html
[3] https://pymoo.org/algorithms/moo/age.html

2.2. Related work

White-box testing [84,117,135,165,205,231] has been extensively researched, but needs internal model access, which is not always realistic in practice. Therefore, we will adopt a black-box testing approach instead, which purely focuses on modifying a system's input (in our case, an image) to trigger undesired changes in the system's output (in our case, the object classification for the input image). We will employ evolutionary strategies for this; beyond images, these have *e.g.,* been proposed on credit scoring models [71] and speech audio [107,115].

In literature, various black-box attacks on image recognition DNNs have been proposed [132,155,236]. Nguyen et al. [155] generate random images that are noise to humans, but are misclassified as actual objects by a DNN. In our case, we will seek more adversarial examples, where input is kept as close to the original as possible, and thus human-recognizable.

Zhou et al. [236] proposed a hybrid black-box approach that combines EAs with the bisection method. The images are mutated by injecting full black or white pixels. Instead, Chan and Cheng [41] introduced a black-box approach that adds Gaussian noise to large portions of the images. Besides, their work targets object detection models rather than image recognition. In contrast, our paper investigates the adversarial example generation in a multi-objective variant where both (1) the model misclassification and (2) the number of changed pixels are taken into account.

Several works explicitly focused on minimizing perturbations, such that fewer modifications to an image would already lead to different system output. One example of this is the work by Suzuki et al. [198], which proposes a Discrete Cosine Transform-based method for modifying images. While such perturbations parametrically are small, they still will affect many pixels at once. A similar consideration holds for the work by Sun et al. [196], focusing on minimum visibility of the modification from a perceptual perspective, but not explicitly constraining the number of pixels to modify.

On the other end of the extreme, one may search for attacks that modify as few pixels as possible (and as such will naturally not stand out, when compared to the total amount of pixels in an image). For example, Su et al.[194] propose a single-pixel adversarial attack using DE, executed against the classical CIFAR-10[4][121] and ImageNet object classification datasets. Comparing the results on these two datasets, a high success rate is reported for CIFAR-10, but this success rate is much lower for ImageNet, where single-pixel attacks mainly succeed in situations where the model's original classification of the image was already quite low. This may have to do with the difference in search space; the test images in CIFAR-10 are much smaller ($32 \times 32 = 1024$ pixels) than those in ImageNet ($224 \times 224 = 50,176$ pixels).

A stronger, yet compact attack is proposed by Lin et al. [132], who combined DE [193] and the Fast Gradient Sign Method [147] for black-box adversarial sample

[4] https://www.tensorflow.org/datasets/catalog/cifar10

generation. Executing a single-objective attack called *Black-box Momentum Iterative Fast Gradient Sign Method* (BMI-FGSM), to generate an efficient and effective perturbation that is similar to the benign input. Their approach utilizes double-step size and candidate reuse whilst approximating the gradient direction. An initial gradient sign population is generated using DE. The input is then gradually modified using gradient sign approximation until an adversarial example is created that is visibly the same as the original input, but now classified as something different. Lin et al. [132] showed that BMI-FGSM successfully generates adversarial examples for large models, outperforming other state-of-the-art white-box and black-box approaches.

While Lin et al. [132] showed that black-box approaches based on Evolutionary Algorithms (EAs) can be very competitive with their white-box alternatives, existing approaches have various drawbacks. First, BMI-FGSM requires a large number of iterations (in the order of thousands) and population size (hundreds of individuals). In other words, attackers need to query the model under attack many times, increasing the chances of detection. Second, BMI-FGSM combines multiple techniques, making its implementation less trivial and introducing more hyper-parameters to tune. Finally, the generated attacks are not minimal, i.e., the prediction flip is achievable but requires changing all pixels in the original image (or seed).

Deepfool by Moosavi-Dezfooli et al. [147] sits in a grey area, as it discusses gradient approximation in their approach, but does not specifically define a black or white-box approach. Moreover, they do not show many examples of complex image perturbations and focus solely on comparisons between itself and the Fast Gradient Sign Method (FGSM)[77] on the MNIST and CIFAR-10 datasets. Similar to our approach, they attempt to create a minimal adversarial example. However, their approach is effectively a gradient descent algorithm with an adaptive step size. Additionally, when validated against the ILSVRC2012 (the dataset used in this paper), they report an increasingly higher error rate.

With these approaches in mind, we have decided on a black-box approach, which does not require an understanding of the model's inner workings. We simply need to understand the input required and craft it accordingly whilst monitoring the output for discrete changes.

In this paper, we extend our previous work [11], wherein we showed a black-box approach, based purely on DE, could produce adversarial examples with minimal perturbations applied, whilst taking considerably less time than existing methods. We introduce three more multi-objective approaches to compare their effectiveness in generating minimal adversarial examples. Furthermore, we introduce a post-processing approach that reduces the noticeability of our adversarial examples by adjusting the applied changes back towards their original image RGB values.

2.3. APPROACH

Without loss of generality, an image classifier f is a mathematical function/model $f : I \longrightarrow L \times \mathbb{R}^n$, which takes as input an image $i \in I$ and returns a label $l \in L$ and a confidence vector $conf \in \mathbb{R}^n$, which contains the probabilities associated with all labels $l \in L$ in descending order. Here, i is an image of size 224×224 pixels, where each pixel in the image is represented by an RGB integer value in the range 0 to 256. The first element $conf_1$ is the probability associated with the most likely (predicted) label l, while the remaining entries in $conf_2 ... conf_n$ are related to the other possible labels $\in L$.

We can now reformulate adversarial attack generation as a search problem:

Definition 1. Let $f : I \longrightarrow L \times \mathbb{R}^n$ be a trained model that takes as input an image $i \in I$ and returns a predicted label $l \in L$ and a confidence vector $conf \in \mathbb{R}^n$. Let $m : I \longrightarrow I$ be a transformation function that mutates (i.e. applies changes to) an image $i \in I$. The problem is finding a mutated image $m(i)$ such that $f(m(i)) \neq f(i)$, with the constraints that both i and $m(i)$ share the same correct label (same ground truth/oracle).

Attack generation strategies can be *targeted* or *un-targeted*. The former aims to flip the prediction to a specific label or classification outcome, while the latter aims to lead the model toward producing any incorrect outcome.

In this paper, we focus on un-targeted attack generation: for demonstrating the vulnerability of a machine learning model, it is sufficient to generate mutated images $m(i)$ that flip the predicted output to any other label than the ground truth label. Since a classification model returns both the label and the corresponding confidence level, we can use the latter to guide the search toward the flipped prediction. More precisely, given a classification model $f : I \longrightarrow L \times \mathbb{R}^n$, a seed image i, and its mutated variant $m(i)$, we optimize the following objective:

$$\min O_1 = \begin{cases} f(m(i))_1^{conf} - f(m(i))_2^{conf} & \text{if } f(i)_1^l = f(m(i))_1^l \\ -f(m(i))_1^{conf} & \text{if } f(i)_1^l \neq f(m(i))_1^l \end{cases} \tag{2.1}$$

In other words, this objective aims to reduce the confidence for the most likely prediction/label ($f(m(i))_1^{conf}$), while increasing the confidence for the second-most-likely prediction ($f(m(i))_2^{conf}$). Therefore, the overall goal is to reduce the difference between the top-2 labels until the model f flips the prediction to a different label (condition $f(m(i))_1^{label} \neq f(m(i))_1^{label}$). In general, Equation 2.1 takes values in [-1,1]. A zero value indicates that the models assign equal confidence scores to the top-2 labels. A negative value indicates that the model f flips the prediction to a different label, whose confidence level corresponds to the absolute value of Equation 2.1.

We can expand this to a multi-objective problem where both "fooling" the model and reducing the number of perturbations (at the pixel level) are equally important:

Problem Definition 1. The problem is finding a mutated image $m(i)$ such that $f(m(i)) \neq f(i)$ and that minimizes the distance $d(i, m(i))$, with the constraints that both i and $m(i)$ share the same correct label (same ground truth).

Beyond flipping the prediction outcome by optimizing for O_1, we now also need an additional objective to guide the search towards minimizing the difference between the original image i and its mutated counterpart $m(i)$.

Therefore, our second objective counts the number of pixels that differ between the seed image i and the mutated image $m(i)$:

$$\begin{aligned} \min\ O_2 &= \pi(m(i[a,b]) \neq i[a,b]) \\ &= |\{i[a,b] \in i : i[a,b] \neq m\,(i[a,b])\}| \end{aligned} \tag{2.2}$$

where $i[a,b]$ and $m\,(i[a,b])$ denote the pixel values in row a and column b for the two images i and $m(i)$, respectively.

These two objectives are *conflicting*. A simple solution for O_1 may include changing all pixels in the original figure i so that the object is no longer recognizable for the model f. However, such a solution would not be optimal for O_2. Vice versa, a new image with zero alteration would be optimal for O_2 but not flip the prediction as sought by O_1.

Given our objectives' conflicting nature, it is impossible to find one single solution that optimizes them all. In other words, the problem is inherently multi-objective, and the goal is to find the set of optimal trade-offs between O_1 and O_2.

In our context, a solution (image perturbation) x is said to *dominate* another solution y (denoted with $x <_p y$) if and only if (1) x is better than y *w.r.t.* to one of the objective (*e.g.,* O_1) and it is not worse than y for the other objectives. Among all possible solutions, we are interested in finding a set of solutions x^* that are not dominated by any other solution:

$$O(x^*) <_p O(y) \ \ \forall y \neq x^* \in \Omega(i) \tag{2.3}$$

where $\Omega(i)$ denotes the set of all possible perturbations that can be applied to the seed image i. All the solutions that are not dominated by any other solutions are said to form a *Pareto optimal set*. The corresponding objective vectors (containing the values of two objectives O_1 and O_2) are said to form a *Pareto front* (or frontier).

2.3.1. Single and multi-objective differential evolution

To find adversarial attacks, we apply our differential evolution (DE) approach in two ways. Firstly, we apply `GA`$_{\text{DE}}$, a traditional single-objective variant (to optimize O_1). Secondly, we apply multi-objective variants (`NSGA-II`$_{\text{DE}}$, `NSGA-III`$_{\text{DE}}$, `MOEA/D`$_{\text{DE}}$, `AGE-MOEA`$_{\text{DE}}$) to optimize O_1 and O_2. All our variants are based on their respective traditional *environmental selection* operators, which are detailed below.

All variants iteratively evolve a pool of N randomly generated adversarial attacks, called *population*. In each iteration, N offspring attacks are generated from the population using variation operators. Then, the population for the next iteration is obtained by combining the previous population and the offspring attack, forming a pool Q of $2 \times N$ attacks and selecting the N top individuals. The *environmental selection* of these individuals is specific to each algorithm and represents the main difference between the single- and multi-objective problems.

- In `GA`$_{\text{DE}}$ [97] (single-objective), the *environmental selection* is applied by selecting the best N individuals among the parent and the offspring solutions/attacks according to the main objective O_1. This mechanism is *elitist* since the best attacks can survive across the generations until new better solutions are found.

- In `NSGA-II`$_{\text{DE}}$, the *environmental selection* is performed by applying the *fast non-dominated sorting algorithm* NSGA-II [57], which ranks the solutions in Q into sub-dominated fronts based on the dominance relation. For solutions within the same front, the selection is further made considering the *crowding distance*, which aims to promote solutions in the less-crowded areas of the front.

- `MOEA/D`$_{\text{DE}}$ [232] is a decomposition-based evolutionary algorithm [232] that breaks down multi- or many-objective problems into several single-objective sub-problems through sum scalarization. It initializes a set of predetermined search directions (or reference vectors) evenly spread across objective space. These directions are obtained by normalizing the objective scores of the individual in the population and applying systematic approaches (*e.g.,* [54]) to create uniformly distributed directions, each with different weights. During the search, `MOEA/D`$_{\text{DE}}$ promotes solutions that are closer to those reference directions, optimizing for both optimality and diversity.

- `NSGA-III`$_{\text{DE}}$ [56] is a many-objective extension to NSGA-II. Like its predecessor, `NSGA-III`$_{\text{DE}}$ uses the non-dominated sorting algorithm. Next, it utilizes the *reference points* defined in the starting parameters, to find a solution with the smallest perpendicular distance to the reference directions in the normalized objective space. The reference points are generated using the Das and Dennis systematic approach [54], which are uniformly distributed in the objective space, thus promoting diversity in the generated solutions.

- `AGE-MOEA`$_{\text{DE}}$ [160], the algorithm executes similarly to *NSGA-II*, but with a modified *crowding distance* formula. Non-dominated fronts are sorted using the non-dominated sorting procedure (like in NSGA-II), whereby the first front is utilized to normalize the object scores and estimate Pareto front geometry. This geometry (or curvature p) is estimated using heuristics and used to measure both the diversity and proximity of the generated solutions using the p-norm.

In the following, we detail (1) the encoding schema, (2) how we initialize the initial population, and (3) the variation operator.

2.3.1.1. ENCODING SCHEMA

As mentioned before, an adversarial attack is produced by altering a seed image i. Instead of representing/encoding an adversarial attack as a completely new image, we only encode the changes to be applied, also called the *mask*. In particular, given the seed image i, we encode a solution/attack as a list of pixels to change: $X = [x_1, ..., x_k]$. Each entry x_j in X is a tuple $[a, b, value_j]$, where a and b determine the position of the pixel to change (i.e., a is the row index and b is the column index), while $value_j$ indicates the new pixel value in RGB notation.

2.3.1.2. INITIALIZATION

We first begin our initialization by generating an initial pool of adversarial attacks. To this aim, we create N attacks by creating an empty mask $X = []$ and adding some changes using the *add* operator, one of three alternative variation operators described below.

2.3.1.3. VARIATION OPERATOR

Given a parent attack X, we design three types of operators that *add*, *delete*, or *change* entries in X. Each operator is applied with probability 1/3.

The **add** operator randomly inserts one entry in X with probability $P = 1$; a second entry is added with probability $P = 0.5$; the third one with probability $P = 0.25$; and so on until no other element is added. To add a new element/entry x in X, this operator randomly selects one pixel from the original seed image i with position row_j and col_j and draws three random (noise) values $\delta(\mu, \lambda)$ from a Gaussian distribution with mean μ and standard deviation λ, that will be applied to the respective R, G and B channels. Hence, the new entry x will be equal to $\big[row_j, col_j, value_j + \delta(\mu, \lambda)\big]$.

The **delete** operator simply deletes one entry/tuple from the mask X. However, this operator is applied only if X contains at least two entries. This operator plays a critical role in our multi-objective formulation as it allows to remove spurious pixel changes that do not contribute to changing the prediction results of the model f under test.

The **change** operator selects an entry from the mask X and modifies the pixels values using the traditional *differential operator*. Given a parent mask to mutate X and a donor mask Y, a new solution X' is formed by using the following formula:

$$X'[a,b] = \begin{cases} R_1[a,b] + F \cdot (R_2[a,b] - Y[a,b]) & \text{if } r < CR \\ X[a,b] & otherwise \end{cases} \tag{2.4}$$

where $r \in [0, 1]$ is a randomly generated number; R_1 and R_2 are other solutions within the population. There are various variants of the differential operator that differ in how R_1, R_2, and the donor solution Y are selected. In this paper, we use the standard `DE/rand/1` variant, where `rand` indicates that the donor Y is randomly selected, while `1` indicates there is only one donor solution. Finally, the other solutions R_1 and R_2 are always randomly selected from the population.

In Equation 2.4, $F \in [0, 2]$ is called *scaling factor* and has a typical value in the range 0.5 - 1.0. It establishes the amplification of the new solution X' from the original solution X based on the differential values of the donor solution Y. Hence, F balances both exploration and exploitation. Finally, $CR \in [0, 1]$ is the *crossover rate* and determines how many pixels in X will be changed.

If the pixel $X[a, b]$ differs from the original seed solution, Equation 2.4 may remove this change if $R_1[a, b]$, $R_2[a, b]$, and $Y[a, b]$ are identical to the original seed image. To prevent this case, we apply a small tweak in our context compared to the traditional differential operator. We set the pixel $R_1[a, b] = [0, 0, 0]$ (in RGB notation) if $R_1[a, b]$ is identical to the pixel of the initial image/seed. The same is done for $R_2[a, b]$ and $Y[a, b]$. Notice that this tweak is applied only if $X[a, b]$ differs from the original seed's pixel in row a and column b.

2.3.2. POST PROCESSING

To post-process the generated adversarial attacks, we take a previously generated mask X and modify the contents, whilst maintaining a flipped prediction. Each element (x_j) in X is a tuple $[a, b, value_j]$, where a and b represent the location of the pixel in the original image, and $value_j$ signifies the color applied to that pixel.

Our process involves selecting a pixel from X (x_j) and gradually mutating $value_j$ towards the original pixels values, whilst ensuring that our prediction remains flipped. If, during this color mutation process, $value_j$ reverts to its original value, x_j is removed from X, thus reducing the number of changes we apply to the adversarial example. The color mutation is applied in a loop, increasing 20% at a time. If, at any point during the loop, the prediction becomes un-flipped, we then revert $value_j$ to the previous working value in the loop and seek a new $x_j \in X$ to minimize.

To ensure a suitable time for post-processing to further minimize our adversarial examples, we run a loop with an arbitrarily chosen $100,000$ iterations. The loop will continue until it reaches the total number of iterations. We also apply a `stop_counter` variable. This has an initial value of 30, which gradually decreases on failed pixel mutations and is reset upon successful pixel mutations. This process is detailed in Algorithm 2.1.

2.4. STUDY DESIGN

To answer RQ1, we run each of our five approaches against five different, well-known, deep-learning computer vision models from the `Keras` python library. We chose

Algorithm 2.1 Post-processing matrix mutation

Require: Y = original image from ImageNet ILSVRC2012 validation dataset
Require: X = change matrix from RQ1
Require: $Y^* = Y$ with change matrix applied
Ensure: $X^* \subset X$

$X^* \leftarrow X$
$max_iterations \leftarrow 100,000$
$stop_counter \leftarrow 30$
for $i = 1$ to $max_iterations$ **do**
    $x_j \leftarrow$ randomly selected item from matrix X^*
    while $prediction(Y) \neq prediction(Y^*)$ **do**
        $original_pixel \leftarrow Y[x_j][a, b]$
        $current_pixel \leftarrow X^*[x_j]$
        $diff \leftarrow Y[x_j][value_j]/100 \times 20$ ▷ Take 20% of the original pixels colour value
        $Y^*[x_j][value_j] \leftarrow Y^*[x_j][value_j] + diff$
        if $prediction(Y) = prediction(Y^*)$ **then**
            $stop_counter \leftarrow stop_counter - 1$
            if $stop_counter = 0$ **then**
                end ▷ Stop the algorithm
            end if
            $Y^*[x_j] \leftarrow current_pixel$ ▷ Resets pixel to before we started modifying it
            break
        end if
        if $Y^*[x_j][value_j] \geq original_pixel[value_j]$ **then**
            $Y^*[x_j] \leftarrow original_pixel$
            $X^*.remove(x_j)$
            break
        end if
        if $prediction(Y) \neq prediction(Y^*)$ **then**
            $stop_counter \leftarrow 30$ ▷ Reset stop_counter
        end if
    end while
end for

`VGG-16` and `VGG-19` which are both classified as "very deep" convolutional neural networks for large-scale image recognition[5] [184], that got a canonical status due to their strong performance in the `ImageNet` benchmark challenges[6]. `VGG-16` was one of the best-performing models in the 2014 ILSVRC challenge and achieved 92.7% top-5 test accuracy on the ImageNet dataset. `VGG-16` and `VGG-19` both consist of 3x3 convolutional layers stacked on top of each other in increasing depth, with `VGG-16`

[5]https://keras.io/api/applications/vgg/
[6]https://image-net.org/challenges/LSVRC/index.php

having 16 convolutional layers, and `VGG-19` being 'deeper' with 19 convolutional layers. Furthermore, `ResNet50`, `ResNet101` and `ResNet152` all are based on deep residual learning for image recognition[7] [91]. As input, the models take images of size $224 \times 224 \times 3$ pixels, with L (the collection of possible target labels) encompassing 1000 different object classes.

In RQ1, we ran each image, model, and approach combination a total of 10 times. This resulted in $\approx 37,500$ test runs in total, the majority of which would generally result in the creation of an adversarial example. To now answer RQ2, we sample from this set of test runs, randomly selecting an adversarial example from each of the image, model, and approach combinations. It should be noted that post-processing was not executed against all combinations due to an adversarial not always being found during RQ1. With our adversarials selected, we execute post-processing 10 times against each adversarial image to create an aggregated set of results.

In our experiments, we use the ImageNet pre-trained weights released by the original authors after training on the ILSVRC2012 training set, as released through `Keras` [46]. Note that despite these details, we consider all models as a black box, given the fact that our approach (and the baseline) does not need access to the model internals.

2.4.1. Dataset

For our experiments, we sample 150 images from the ImageNet ILSVRC2012 validation dataset [174]. As such, this data contains verified ground truth labels, that were obtained in similar fashion to the labels on which the models under test were trained. Thus, in choosing to keep sampling from the ILSVRC2012 validation dataset, we are certain that this data should not be out-of-distribution for the model, while at the same time can assume the models have not yet seen our testing data during training time.

To ensure unbiased pre-formatting, we perform the same steps to resize images during dataset acquisition as when running our main experiment. By utilizing the `Pillow`[8] Python library, we resize using bi-linear interpolation. This ensures samples are resized as close to the original as possible.

As for the prediction confidence, if the correct ground-truth label is predicted with confidence between 0.8 and 0.9, we consider the image to be a valid image for our experiments: as the ground-truth label is recognized with high confidence, we can be confident it is visually distinguishable and not ambiguous *w.r.t.* other classes. Concerning this, an image with too high confidence (*e.g.,* greater than 0.9) may be too obviously one particular image class, and flipping may be difficult as a consequence. By retaining only one image per object class, we also ensure a reasonably diverse set of images.

[7] https://keras.io/api/applications/resnet
[8] https://pypi.org/project/Pillow/

Algorithm 2.2 Dataset Acquisition

Require: Ω = A set of 1000 randomly selected images from ImageNet ILSVRC2012 validation dataset
Require: f = the model under test
Ensure: $\Omega^* \subset \Omega$
1: $Classes = \emptyset$
2: $\Omega^* = \emptyset$
3: **for** $\omega \in \Omega$ **do**
4: **while** $|\Omega^*| < 150$ **do**
5: $i \leftarrow convert(\omega, 224 \times 224)$
6: **if** $0.8 \leq f(i)_1^{conf} \leq 0.9$ **then**
7: **if** $f(i)_1^{label} \notin Classes$ **then**
8: **if** $f(i)_1^{label} \equiv GroundTruth(i)$ **then**
9: $\Omega^* = \Omega^* \cup \{i\}$
10: $Classes = Classes \cup \{f(i)_1^{label}\}$
11: **end if**
12: **end if**
13: **end if**
14: **end while**
15: **end for**

Algorithm 2.2 helps to outline this information. We first draw an initial pool of images for the input by selecting 1000 random images from the ImageNet validation data folder. After pre-processing the images to have a 224×224 input size, (line 5 in Algorithm 2.2) we then choose whether to drop or retain the image, based on the prediction confidence by the `VGG-19` model (condition in line 6) and class uniqueness (condition in line 7).

2.4.2. Implementation and Parameter Settings

We have implemented the various DE-based approaches in `Pymoo` v0.6.0 [32], using Python 3.10 and `Keras` 2.12.2. Pymoo [32] is an open-source framework that allows us to easily adapt the original search algorithms to our DE approach. Runs are evaluated inside a Docker container on an AMD EPYC 7713 64-Core Processor running at 2.6 Ghz and with 256 available CPUs. We had 3 Nvidia A40 GPUs each with 48 GB GDDR6 running CUDA version 11.6 available to us. The Dockerfile in our implementation can be rebuilt on any system, easily modifying the CUDA container for a different system.

2.4.2.1. Parameter Settings

We set our approaches, both single- and multi-objective DE, to evolve a population size of 20 over a maximum of 600 generations. In the case of `MOEA/D`$_{\text{DE}}$ and

NSGA-III$_{DE}$, we were also required to supply reference directions. For MOEA/D$_{DE}$, we used Das and Dennis's systematic approach [54] for generating well-spaced reference points, as done in the original paper [56]. For MOEA/D$_{DE}$, we use uniform sampling on the unit simplex. For both algorithms, we set the number of partitions (`n_partitions`) equal to 20, i.e., identical to the population size. For the parameter settings, we have chosen the same values as suggested in the literature [57,146]. Further to this, when an adversarial example is found, we allow the test to run for a further 20 generations before killing it and returning the results. This allows the test to continue the search for a better adversarial example, if one may exist.

We use the variation operators as described in Section 2.3 with a crossover rate $CR = 0.9$ and scaling factor $F = 0.8$, which are the recommended values in the literature [146]. For all algorithms, solutions/attacks are selected for reproduction using the binary-tournament selection [57]. For GA$_{DE}$, the binary selection is based on the single-objective value to optimize (i.e., O_1). Instead, in the multi-objective approaches, the selection relies on dominance to decide which solution wins each tournament round. Finally, we opted for a relatively small population size $p = 20$ (smaller than p=100 used in other studies [132,146]) as suggested in the literature from problems with expensive objective computation [47].

2.4.3. Evaluation criteria

In our previous work [11], we concluded that the single-objective approach GA$_{DE}$ was able to create adversarial examples quicker than our multi-objective NSGA-II$_{DE}$. However, NSGA-II$_{DE}$ would create adversarial examples with fewer perturbations. To answer our first research question, we analyze our multi-objective approaches to see how they compare against each other. As mentioned earlier in the paper, we utilized AGE-MOEA$_{DE}$ and NSGA-III$_{DE}$ as multi-objective problems.

We execute GA$_{DE}$, NSGA-II$_{DE}$, NSGA-III$_{DE}$, MOEA/D$_{DE}$ and AGE-MOEA$_{DE}$ against each image in our dataset. For every image, we run each algorithm 10 times, to account for their random nature. As a consequence, with 150 images, the end result is a total of 37,500 test runs (7500 for each method). For each of the executions, a new random seed was generated and stored for future replications, together with the results of the generated attacks.

For the main evaluation, we consider two performance metrics: (1) the success rate, indicating the percentage out of the 10 runs for which our methods were capable of causing a change in prediction output, and (2) how many pixels needed modification (i.e., how many tuples are in mask X) in the best solution. For the comparison, we considered the best solution/attack. With GA$_{DE}$, this is the attack with the smallest number of perturbations (changed pixels). Our multi-objective solutions, however, provide a set of Pareto-optimal solutions. For our analysis, among all solutions/attacks that lead to flipping the prediction (i.e., those with negative values for the first objective O_1), we have chosen the one with the lowest number of changed pixels (second objective O_2).

To further visualize the effectiveness of our approaches, we also consider the run-time, along with the number of generations required for generating an adversarial example. As stated in the Parameter Settings (Section 2.4.2.1), we allow a total of 600 generations for each test run. On top of this, we also record the run-time. This is a simple calculation, taking the time between the start and end of a single test run.

To assess the significance of the differences among our approaches, we use the Friedman test [50], following the guidelines [61,69,162] for comparing meta-heuristics across a collection of independent benchmark problems (images in our context). We employed the Friedman test —a non-parametric counterpart to ANOVA— because it is suitable for unreplicated block designs and it does not require data to adhere strictly to a normal distribution. For this test, we use the confidence level $\alpha = 0.95$.

While the Friedman test reveals whether some approaches (DE variants) statistically differ, it does not pinpoint the specific combinations (DE variants) for which such a significant difference holds. To gain a deeper understanding of which approach(es) outperform, we utilized the Nemenyi [154] post-hoc test. This test assesses treatment disparities by determining the average rank of each DE variant across multiple problems [154] (images in our case). The significance between two DE variants (*e.g.,* `NSGA-III`$_{\text{DE}}$ and `MOEA/D`$_{\text{DE}}$) is established if their respective average ranks differ by at least the provided *critical distance* (CD) [154,162].

To evaluate the effectiveness of our post-processing approach, we observe two performance metrics. (1) the overall color difference applied by the original adversarial example compared to the post-process adversarial and (2) the number of elements in the mask. For the comparison, we obtain the color difference by utilizing the `OpenCV`[9] Python library. The *absdiff* method from `OpenCV` takes two images as its input, to output the overall RGB channel difference between the two images. This gives us two values, *start_diff*, which is the difference between the original image and the adversarial example. As well as *end_diff*, which is the difference between the original image and the post-process adversarial. Finally, we compare the size of the original adversarial examples mask (*start_mask*) to the post-process adversarial mask (*end_mask*).

2.5. Empirical Evaluation

In this Section, we discuss the results of our research questions separately.

2.5.1. Results for RQ1

Figure 2.1 compares all five approaches with regard to their ability to flip a model's prediction. The box plot represents the success rate for the total number of executions. Where each image, model, and approach was executed 10 times. As

[9] https://github.com/opencv/opencv-python

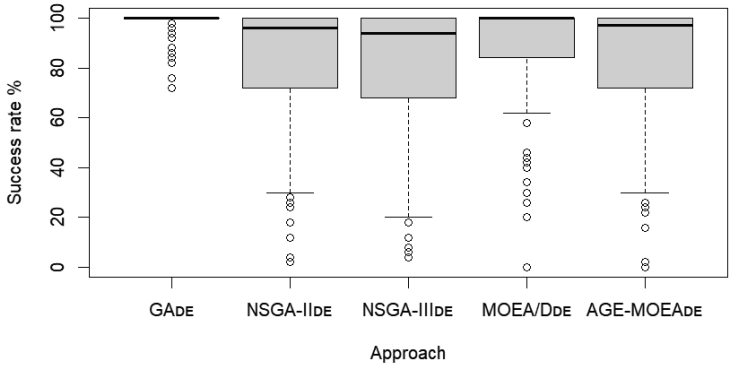


Figure 2.1. Success rate of each approach for all 150 images over all 5 models.

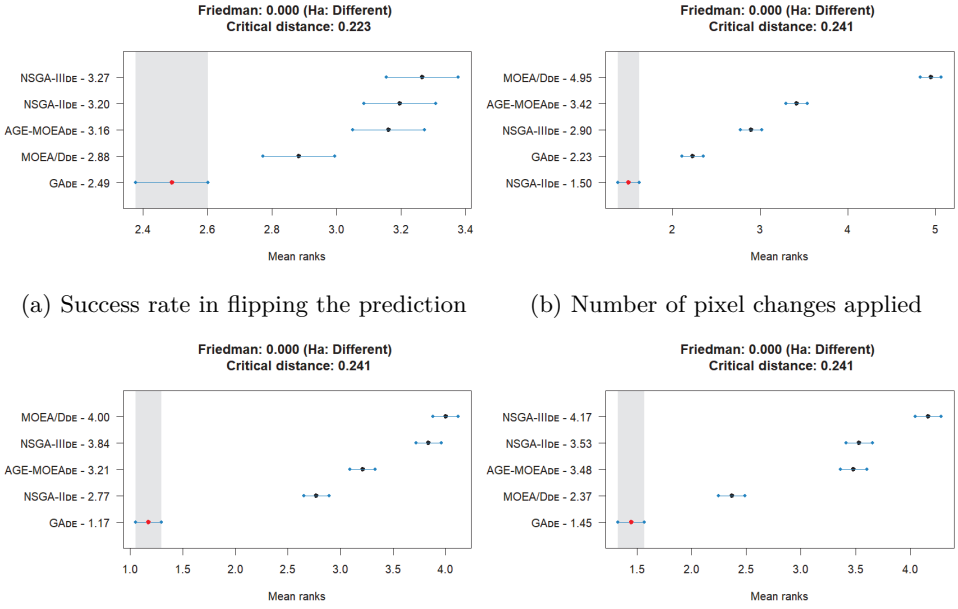
expected from our previous work, we see that GADE is extremely effective at flipping the prediction of an image.

To better understand the results of our experiment, we turn now to Figure 2.2, which displays the results of the Friedman test and of the Nemenyi post-hoc procedure. These statistical tests confirm the results shown in Figure 2.1, as we see statistically significant superiority of GADE in flipping an image's prediction, i.e., generating a successful adversarial attack. The Friedman test reveals a significant p-value < 0.05, while Nemenyi test suggests that the single-objective DE variant is statistically better than all other meta-heuristics in the comparison.

Interestingly, we observe that MOEA/DDE is a strong contender when flipping a model prediction. This is evident in Figure 2.1 wherein the median success rate for MOEA/DDE and GADE are identical (100% success rate), with the former showing only a lower first and second quartile. However, whilst we observe that MOEA/DDE performs well during prediction flips, we also note that it is less successful in other aspects of our test.

For example, Figure 2.2b displays MOEA/DDE as significantly worse than all other approaches *w.r.t.* to the number of pixel changes applied. This is also evident when looking at Figure 2.3, in which we visibly see the number of changes applied by the various approaches. Furthermore, we also observe a statistically better performance of NSGA-IIDE *w.r.t.* its ability to apply minimal (i.e., fewer) pixel changes.

When we consider the run-time along with the number of generations (our maximum being 600), we notice that GADE performs considerably well. It is also worth noticing that MOEA/DDE requires markedly fewer generations compared to the other multi-objective approaches. However, it has a high run-time associated with this. As a



(a) Success rate in flipping the prediction

(b) Number of pixel changes applied

(c) Run-time required to generate adversarial examples

(d) Total generations required to generate adversarial example

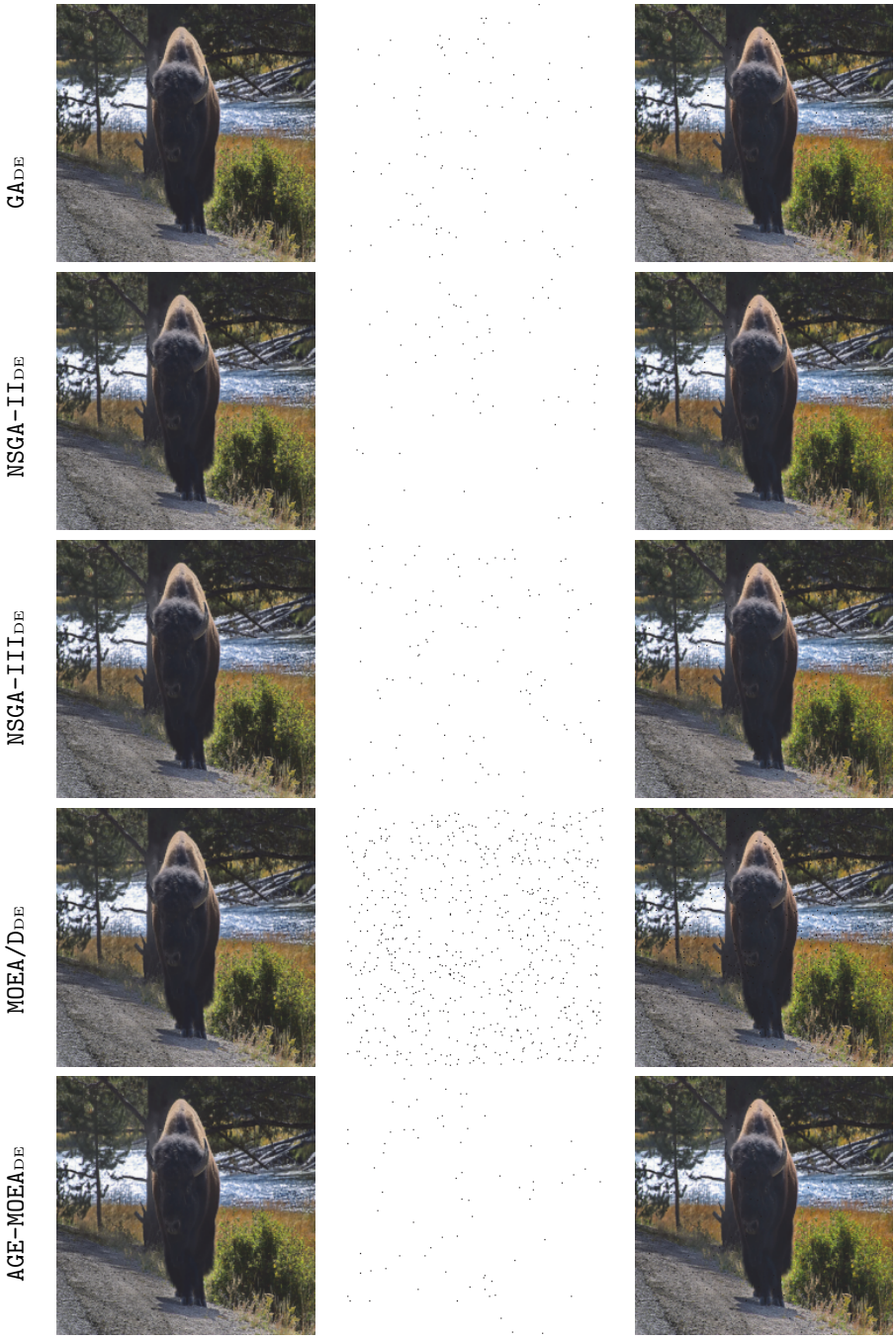
Figure 2.2. Results of the Friedman test and of the Nemenyi post-hoc procedure

result of each iteration of `MOEA/D`$_{DE}$ being more expensive than for other DE variants due to the computational cost of its internal environmental selection strategy.

In order to comprehend the impact of these outcomes, we will now conduct an in-depth look at the `VGG-19` specific results. We have chosen `VGG-19`, due to it having been the baseline for image selection, as mentioned earlier in Section 2.4.1.

To better understand the type of attacks generated by our multi-objective approaches, Figure 2.3 shows the results of a randomly chosen image in our dataset ('ILSVRC2012_val_00030959.JPEG'). This image was successfully flipped by all five approaches and was originally ground-truthed as 'bison'(165), with a confidence value of 0.8631084. Here, the confidence value represents the model's prediction confidence, where 1.0 is the highest obtainable value (i.e., the model is 100% certain about the prediction).

With Figure 2.3, we are able to witness the transition from the original image, to the final image with the mask applied. As previously demonstrated in Figure 2.2b, we can observe that `NSGA-II`$_{DE}$ generated the smallest mask size, with `MOEA/D`$_{DE}$ having a considerably larger mask. This is increasingly apparent in the final image for each approach, where the changes applied by `MOEA/D`$_{DE}$ are more pronounced.



(a) Original image **(b) Mask** **(c) Final image**

Figure 2.3. Adversarial example generation results for Image `ILSVRC2012\_val\_00030959.JPEG`

Table 2.1. Original adversarial example (start) and post-process adversarial (end) execution data for ILSVRC2012_val_00022868

Approach	start diff	end diff	start mask	end mask	run-time	reps
GA$_{DE}$	7438	1086	28	3	34	85
NSGA-II$_{DE}$	4719	549	16	2	18	47
NSGA-III$_{DE}$	4875	1268	20	5	19	48
AGE-MOEA$_{DE}$	12884	2692	48	9	44	102
MOEA/D$_{DE}$	1550187	415	6098	3	4893	7739

Finally, given that our goal is minimal adversarial example generation, we observe the overall applied changes by each approach. Figure 2.4 displays each model tested, along with the changes applied by each approach over the 150 test images. This box plot affirms our previous observations, in which NSGA-II$_{DE}$ generated the least number of changes in an adversarial example, followed by GA$_{DE}$. MOEA/D$_{DE}$ generated the largest number of changes, but with quite a broad distribution. While AGE-MOEA$_{DE}$ and NSGA-III$_{DE}$ are our middle ground for applied changes.

2.5.2. Results for RQ2

When investigating the masks generated by our approaches, it is visible in Figure 2.3 that they tend to work towards a dark-colored pixel $RGB(0, 0, 0)$. In some cases, this can lead to less subtle adversarial examples. Therefore, we applied post-processing to our adversarial examples to observe whether this technique could prove useful in the minimal attack generation process.

Table 2.1 displays the results of our post-processing against the 'Welsh springer spaniel' image (ILSVRC2012_val_00022868). Here we witness the *start* and *end* specific values of color difference and mask size. These values for our original adversarial example, along with our post-process adversarial, allow us to observe the reduction obtained by running post-processing. Lastly, we note the post-processing time required in reaching this further minimized adversarial example, along with the number of repetitions taken.

What we witness from these results, is that post-processing can drastically reduce the amount of change in our adversarial examples. More importantly, for MOEA/D$_{DE}$, which, as mentioned previously generated an extremely large number of changes, post-processing yields a considerably smaller number of changes, rivaling that of other approaches.

NSGA-II$_{DE}$ continues to be the best-performing model *w.r.t.* minimal perturbation; however, this is likely due to the size of the original mask. A smaller mask results in a smaller search space for the post-processing approach, leading to fewer repetitions and quicker post-processing. It is interesting, that despite the smaller mask size of

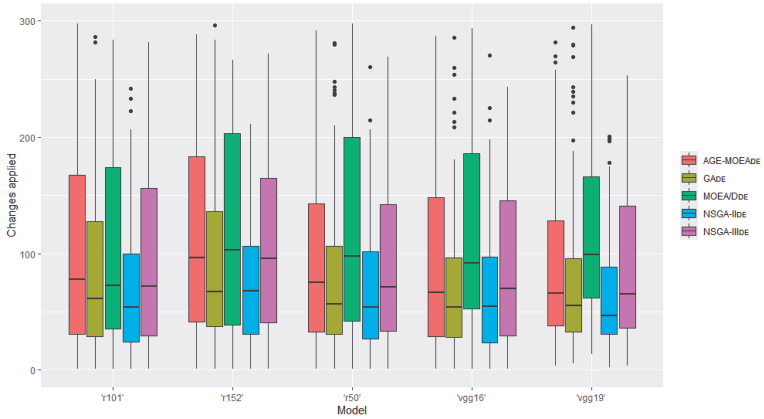


Figure 2.4. Changes applied by each approach to each model

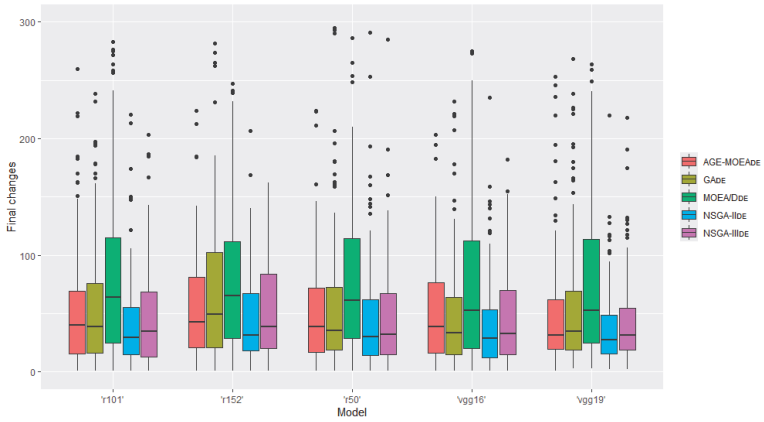


Figure 2.5. Changes applied after post-processing

NSGA-IIDE, we are still able to observe a reduction of nearly half the original changes. For example, by obtaining the overall reduction percentage between *start_diff* and *end_diff*:

$$\text{reduction} = \frac{\text{end_diff} - \text{start_diff}}{\text{start_diff}} \times 100 \tag{2.5}$$

We ascertain a reduction of 42.86% over the color our original adversarial example applied versus the post-process adversarial. By utilizing the same Equation 2.5, but for *start/end_mask*, we produce a mask reduction of 40%.

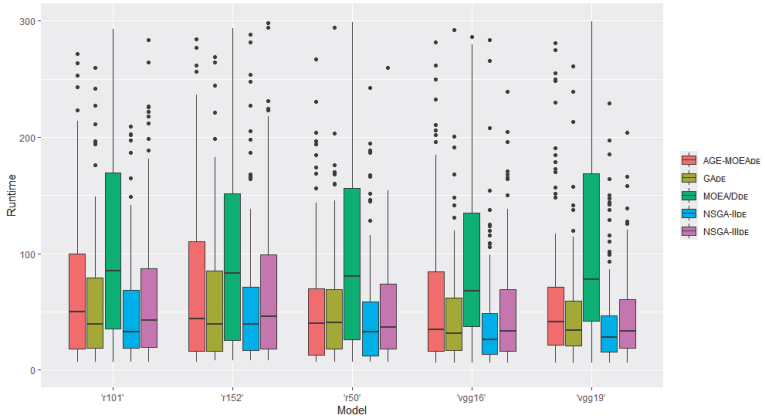


Figure 2.6. Median post-processing run-time

Further evidence can be observed through Figure 2.5, which displays the median mask size after post-processing was applied. Here, we take the median values for the 10 runs of each image, model, and approach combination, where an original adversarial was successfully generated.

To complement this, we also take note of Figure 2.6. Here, we observe the median run-time for the same examples as Figure 2.5. As displayed in Table 2.1, run-time is heavily correlated to the original mask size.

Finally, we observe in Figure 2.7 the overall mask reduction, together with the original and final adversarial examples. We display here the changes from the `MOEA/D`DE approach against the 'ILSVRC2012_val_00022868' (Welsh springer spaniel) in the `ResNet101` model.

2.6. Threats to validity

Several threats to validity can be identified for our study. In the next subsections, we discuss the threats and how we addressed them.

2.6.1. Construct validity

While we assume our adversarial examples have perceptually visible changes, but these changes are small enough not to change the object of focus (i.e., the ground truth) for an image, we do not formally validate this in human experiments. Thus, it is possible that found adversarial examples may be degraded to the extent that a change in ground truth label would be needed and justifiable. However, in our current experiments, we already see many prediction flips happening when changing less than 200 (so less than 0.3%) out of 50,176 pixels, making it unlikely that the object of focus would be completely obscured by the mutations. Next to this, we

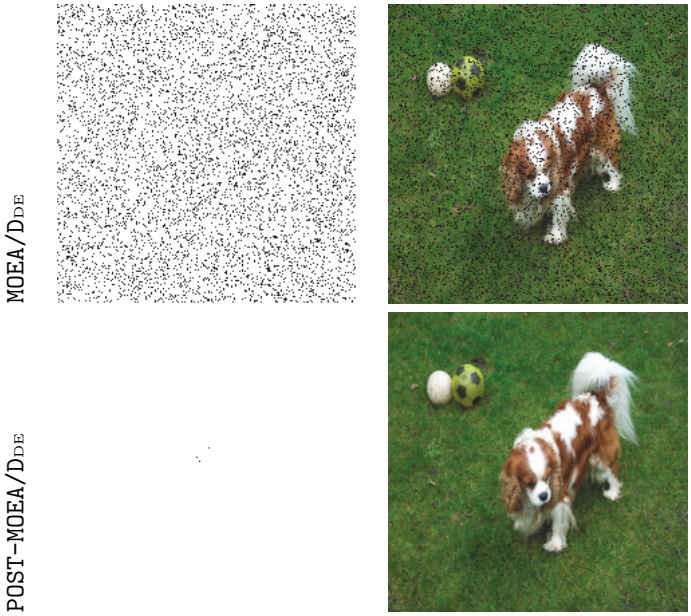


Figure 2.7. Post-processing reduction for Image 'ILSVRC2012_val_00022868'

validated the model's initial ground-truth label against the data supplied by the ImageNet website. Despite this, the model may have an erroneous initial prediction, and even if it would be correct in comparison to the ground truth, plausible labels beyond the indicated ground truth may exist if the visual scene is complex or semantically ambiguous [131]. Thus, while we frame our technique as one creating adversarial attacks, it cannot be guaranteed that we necessarily push the model towards being wrong. Still, the aspect about adversarial attacks that keeps holding is that subtle changes lead to different prediction outcomes; as such, a flip in output prediction is still an indicator of the model not giving robust predictions.

2.6.2. Internal validity

While our image selection procedure yielded a random draw of images from 150 unique classes, the ILSVRC2012 classes semantically are not uniformly distributed (*e.g.,* having multiple classes with sub-species of dogs). Future sampling strategies could seek to do more explicitly mitigate this.

Furthermore, as noted in Section 2.4.1, our image selection remained limited to the ImageNet ILSVRC2012 validation dataset. Although this may seem like a limiting choice, we could assume the models under test had not seen the images during training. Coupled with this, the images closely aligned to the training data, in both

acquisition and ground-truth labeling procedure. As such, they should not be out-of-distribution for the models under test, while it is ambiguous whether this may be the case for other, externally obtained images of the learned object categories.

We do recognize that in practical applications, the intention should very likely be that models should robustly be capable of recognizing a broad variety of images exemplifying the object categories they were trained to recognize. As such, for future work, it will be interesting to possibly include such images obtained from other datasets.

However, this will then also require further explicitization of requirements on a model and subsequent performance assertions. If `VGG-19` *e.g.,* should be more than a prediction program trained to recognize exemplars of 1000 object categories that are similar to what was in ImageNet ILSVRC2012, how much more would it be? Should it be considered the same artefact if retrained with more diverse data? Should it be capable of generalizing to object categories that are similar to the 1000 categories it was trained on? But if so, how would this generalization be defined? As we pointed out in earlier work [163], questions like these tend to be under-specified in machine learning contexts, while more explicit articulation will be essential for developing more robust models together with stronger testing mechanisms.

2.6.3. External validity

Currently, our approach was only tested against (the state-of-the-art) `BMI-FGSM` [11]. It will be worthwhile to also test it against further attack approaches, such as the one-pixel attack [194]. Furthermore, beyond our current set of DNN models, more canonical models exist that can be studied, such as Inception-v3 [199].

2.6.4. Conclusion validity

In some runs, our multi-objective approaches fail to find an adversarial example; further optimizations with regard to population and generation size may be required.

2.7. Conclusion and Future Work

In this study, we have introduced three new approaches based on pure differential evolution, along with a post-processing technique. We anticipate that this study will contribute to the efficient generation of concise adversarial examples, facilitating their use in the subsequent training of Deep Neural Networks (DNNs).

The approaches, namely `MOEA/D`$_{\text{DE}}$, `AGE-MOEA`$_{\text{DE}}$, and `NSGA-III`$_{\text{DE}}$, each execute customized differential operators for generating adversarial examples. Our post-processing approach utilizes random search in an image change space to optimize for a less noticeable adversarial example.

Our empirical study with multiple DNN models for image recognition showed that (1) a single-objective approach is the most efficient way of generating adversarial examples. (2) NSGA-II$_{DE}$ is the optimal solution for generating adversarial examples with minimal perturbations. (3) By applying a post-processing approach, we can reduce the overall amount of changes and delta variation for our adversarial examples. This process was effective on all approaches, proving surprisingly effective on MOEA/D$_{DE}$ where the initial amount of changes was considerably higher than others.

In our future work, it will be worthwhile to explicitly look at the potential difficulty of images due to semantic ambiguity, which already can show in the initial classification confidence of a model [131]. Furthermore, we intend to extend our comparison to DNN models with different input/ output criteria.

Moreover, in this study, we utilized many-objective approaches as multi-objective algorithms. In future work, it would be interesting to utilize our many-objective methods with a third objective. As we have seen in this work, we typically reduce our pixel color *(RGB)* value to *black (0,0,0)*. By optimizing the color delta variation (image noise), we could produce better minimal adversarial examples without the need for post-processing.

3

THE PURSUIT OF DIVERSITY: MULTI-OBJECTIVE TESTING OF DEEP REINFORCEMENT LEARNING AGENTS

Testing deep reinforcement learning (DRL) agents in safety-critical domains requires discovering diverse failure scenarios. Existing tools such as `INDAGO` rely on single-objective optimization focused solely on maximizing failure counts, but this does not ensure discovered scenarios are diverse or reveal distinct error types. We introduce `INDAGO-Nexus`, a multi-objective search approach that jointly optimizes for failure likelihood and test scenario diversity using multi-objective evolutionary algorithms with multiple diversity metrics and Pareto front selection strategies. We evaluated `INDAGO-Nexus` on three DRL agents: humanoid walker, self-driving car, and parking agent. On average, `INDAGO-Nexus` discovers up to 83% and 40% more unique failures (test effectiveness) than `INDAGO` in the SDC and Parking scenarios, respectively, while reducing time-to-failure by up to 67% across all agents.

This chapter was awarded best paper and presented at SSBSE'25 co-located with ASE in Seoul Republic of Korea by Antony Bartlett, Cynthia C. S. Liem and Annibale Panichella [15]. See Chapter 1.6 for additional publication details.

3.1. INTRODUCTION

Reinforcement Learning (RL)[197], and more recently Deep Reinforcement Learning (DRL)[144], have emerged as powerful tools for automating complex

tasks in dynamic and uncertain environments [145,182]. DRL agents learn through trial-and-error interactions with their environment and are increasingly deployed in Cyber-Physical Systems (CPS), such as self-driving cars (SDCs) and humanoid robots [34], where safety, reliability, and adaptability are critical.

Ensuring the reliability of DRL-based systems remains a significant challenge [24,209]. Unlike conventional software, DRL agents are opaque, stochastic, and highly sensitive to reward signals. Their behavior depends not only on a learned policy but also on vast environment and action spaces. As a result, edge-case scenarios—those most likely to trigger failures—are difficult to identify, and traditional testing approaches often fall short [209]. Real-world incidents involving autonomous agents [72,80,124] underscore the consequences of failing to detect such rare conditions during testing.

Simulation-based testing is a widely adopted technique for evaluating DRL agents, especially in safety-critical domains such as autonomous agents. It allows developers to generate and execute test scenarios in a controlled and reproducible setting, substantially reducing the time, cost, and safety risks of real-world testing [73,113]. A test scenario refers to a specific configuration of the environment—including initial agent state, obstacle placement, and task-specific parameters—that defines the context in which the agent is evaluated. Despite its advantages, simulation-based testing still remains resource-intensive, and exhaustive exploration of the test input space is often infeasible due to its combinatorial complexity.

To address these challenges, Biagiola and Tonella [24] proposed the use of surrogate models to guide test generation for DRL agents. Their framework, `INDAGO`, builds a surrogate model from training logs to predict whether a given test scenario is likely to cause agent failure—without executing it in simulation. `INDAGO` combines this model with two search strategies: Hill Climbing (HC) and a single-objective Genetic Algorithm (GA). Their empirical evaluation across multiple DRL tasks showed that (1) surrogate models substantially reduced simulation time, and (2) both HC and GA outperformed earlier random-search-based methods [209] in identifying more failure-inducing scenarios.

While `INDAGO` effectively increases the number of failing scenarios, it focuses solely on failure likelihood, overlooking diversity. This is a key limitation—generating more failures does not guarantee that they are distinct. Diversity is crucial in testing as it promotes broader coverage of the agent's operational space and improves the chances of uncovering unique faults, helping to address the *curse of rarity* problem [133].

In this paper, we propose `INDAGO-Nexus`, a multi-objective search framework for DRL testing that jointly optimizes for failure probability and scenario diversity. We investigate:

RQ1 : *How does multi-objective search perform in finding diverse failures?*

RQ2 : *How effective is `INDAGO-Nexus` compared to state-of-the-art `INDAGO`?*

To tackle this inherently multi-objective problem, we employed multi-objective evolutionary algorithms (MOEAs). MOEAs are well-suited for this task as they efficiently explore trade-offs and produce a set of Pareto-optimal solutions [55,86,106]. We experimented with two diversity metrics (Euclidean distance and PCA-based clustering), two MOEAs (`NSGA-II` [57] and `AGE-MOEA` [161]), and two Pareto-front selection strategies (highest failure likelihood and knee point [234]). We benchmarked `INDAGO-Nexus` on three DRL agents: self-driving car, humanoid walker, and parking agent.

Our results indicate `INDAGO-Nexus` discovers up to 83% and 40% more unique failures than `INDAGO` in the SDC and Parking scenarios, respectively, while reducing time-to-failure by up to 67%. Knee-point selection consistently produces the best results across most configurations, while different diversity metrics show varying effectiveness depending on the scenario.

Therefore, our contributions are:

- We propose `INDAGO-Nexus`, a multi-objective approach to generate diverse test environments for DRL agents.
- We demonstrate `INDAGO-Nexus` discovers up to 83% more unique failures than `INDAGO` while reducing time-to-failure by up to 67%.
- We provide a complete replication package with implementation and experimental data[1].

3.2. Background and Related Work

This section describes the key background concepts used in this work and summarizes the related work.

3.2.1. Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) extends Reinforcement Learning (RL) by using deep neural networks as function approximators to handle high-dimensional inputs (*e.g.,* images) and continuous action spaces (*e.g.,* steering angles in autonomous driving) [145]. Instead of explicitly representing policies or value functions, DRL learns them directly from raw observations and rewards using deep learning architectures.

This capability makes DRL particularly suitable for complex tasks in cyber-physical systems such as self-driving cars (SDCs), where the environment is dynamic and the decision space is large. In this work, we focus on *model-free* DRL algorithms, where agents learn directly from interactions without an explicit model of the environment. For example, the DRL agent for the parking scenario we consider in our case study (see Section 3.4) was trained with *Proximal Policy Optimization (PPO)* [180],

[1] 10.6084/m9.figshare.29204687

a widely adopted policy-gradient method known for its stability and sample efficiency.

> *The agent's goal is to maximize the cumulative reward it receives in the long run* [197].

3.2.2. BLACK-BOX TESTING

In black-box testing, diversity is a common strategy for exposing faults by varying inputs and observing outputs [81]. For DRL agents, we focus on input diversity—variations in environment configurations—to guide test generation without costly simulations, combined with a surrogate to estimate failure likelihood.

Related work on diversity-based testing for deep neural networks includes approaches that optimize for input diversity to improve fault detection. Aghababaeyan et al. [4] proposed black-box testing techniques that use test case diversity to generate diverse test inputs for deep neural networks. While their work demonstrates the effectiveness of diversity-based approaches, their diversity metrics are primarily designed for image inputs rather than the tabular data that characterizes our MLP-based surrogate models. Our work adapts the concept of diversity-driven testing to the specific context of DRL environment configurations, which are typically represented as structured tabular data with both continuous and categorical features.

3.2.3. SURROGATES

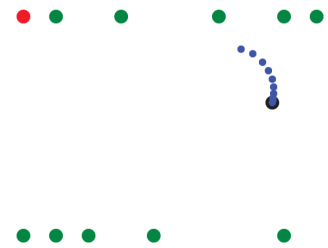
Surrogate models are lightweight machine learning approximations that learn to predict the behavior or outcomes of computationally expensive processes based on historical data. In the context of DRL testing, these models have proven particularly effective at reducing evaluation costs [24]. Specifically, given an environment configuration (Figure 3.1a), a surrogate model can predict the failure probability in the range $[0, 1]$, thereby enabling efficient test generation without requiring costly simulation executions.

3.2.4. TESTING DRL AGENTS

The most directly related work is by Biagiola and Tonella [24], who introduced `INDAGO`, a test generation framework that leverages surrogate models trained on DRL logs to predict whether a given test environment is likely to cause failure. These surrogates return a continuous score in $[0, 1]$, representing the estimated failure probability of an environment. `INDAGO` combines this prediction with search-based strategies, namely Hill Climbing [181] and Genetic Algorithms [75], to efficiently explore the input space.

```
"env_config": {
    "goal_lane_idx": 0,
    "heading_ego": 0.96,
    "parked_vehicles_lane_indices": [
        1, 3, 6, 8, 9,
        10, 11, 12, 14, 18
    ],
    "position_ego": [
        1.83, -4.96
    ]
}
```

(a) Environment configuration



(b) Agent behavior visualization

Figure 3.1. Parking scenario showing (a) environment configuration details used by DRL as a starting point, and (b) the DRL agent behavior. In (b), the red dot indicates the target parking spot, green dots represent parked cars, the black dot marks the agent's starting position, and the blue dots trace its trajectory.

Our work builds on `INDAGO` by introducing a second optimization objective: input diversity. This extension aims to increase the diversity of the discovered failures, thereby improving coverage of the agent's operational space.

3.3. Approach

The goal of our approach is to *generate test scenarios that are both failure-inducing and diverse*, increasing the likelihood of uncovering unique faults in DRL agents. We build upon the `INDAGO` framework [24], which uses a surrogate model trained on DRL training logs to predict whether a given environment is likely to cause a failure. While `INDAGO` focuses solely on failure likelihood, we extend it to a *multi-objective test generation framework* that also accounts for scenario diversity—crucial for uncovering unique faults. Hence, we formulate the problem of generative diverse failure-inducing test scenarios as follows:

Problem Definition 1. Given a DRL agent A, a set of test scenarios E, and a surrogate model $s : E \to [0, 1]$ that estimates the likelihood of failure for a given e, the goal is to generate a set of test scenarios $E_{\text{sel}} \subseteq E$ that maximize the likelihood of A to fail while maximizing diversity among the selected scenarios. More formally, the goal is to find a set of environments E_{sel} that optimizes the following objectives:

$$\begin{cases} \max O_1 = s(e) & \text{(Failure Likelihood)} \\ \max O_2 = Div(e, E_{\text{sel}}) & \text{(Input Diversity)} \end{cases} \tag{3.1}$$

where O_1 (Failure Likelihood) is the predicted likelihood of failure for a given environment e; and O_2 (Input Diversity) is a measure of how diverse the selected

environments are compared to the other test scenarios in E_{sel}.

In the following, we describe the surrogate model, diversity metrics, and the multi-objective optimization framework.

3.3.1. Surrogate Models

Following the prior work by Biagiola and Tonella [24], we adopt a lightweight surrogate model to estimate the failure likelihood of a test scenario without executing it. Specifically, we utilize a multi-layer perceptron (MLP) classifier trained on data collected during the DRL agent's training episodes. The MLP architecture is well-suited for this task due to the relatively small training dataset (up to 10k labeled environments) and the low-dimensional nature of environment feature vectors (*e.g.,* 24-dimensional in the parking scenario). The surrogate outputs $s(e) \in [0, 1]$, representing failure probability for environment e.

3.3.2. Diversity Metrics

We explore two diversity methods: **Euclidean distance** computes the mean distance between a new scenario and previously selected ones, and **PCA-based clustering** [64] measures distance to the nearest cluster centroid in reduced dimensionality space. Both methods handle variable-length features through one-hot encoding and zero-padding.

3.3.3. Multi-Objective Evolutionary Testing

To generate a diverse set of failure-inducing test scenarios, `INDAGO-Nexus` employs MOEAs that evolve environment configurations to simultaneously maximize (1) the likelihood of failure, as predicted by a surrogate model, and (2) diversity with respect to a growing archive of previously selected tests. We use two MOEAs: `NSGA-II` [57] and `AGE-MOEA` [161] to optimize both failure likelihood and diversity.

Algorithm 3.1 outlines this process. The search begins with a population of size PS. Each individual is evaluated without test execution using heuristics described in Sections 3.3.1 and 3.2.2. The algorithm proceeds for G generations. In each generation, two parents are selected (Lines 9-10), the crossover is applied with probability CR (Line 11), and both offspring are mutated (Lines 14-15). Offspring are added to the new population (Line 16), and elitism selects the best individuals for the next generation (Line 18).

To avoid stagnation, `INDAGO-Nexus` monitors hypervolume changes across generations. If improvement falls below a threshold, or we have reached the maximum number of generations G, the best individual is archived (Line 20). To explore new regions of space, the process continues until the total test runs TR is met. The resulting archive E_{sel} contains diverse, likely-failing scenarios, which are then executed on the DRL agent for validation.

Algorithm 3.1 Multi-Objective Test Generation for DRL Agents

Require: s, classifier (surrogate model);
Require: TR, test runs;
Require: G, generations;
Require: PS, population size;
Require: CR, crossover rate;
Require: E_{train}, set of environment configurations where the DRL agent failed during training;
Ensure: E_{sel}, the archive storing a diverse set of likely-failing test scenarios.

```
1: currentTestRun ← 0;
2: while currentTestRun ≤ TR do
3:     population ← GENERATE-POPULATION(PS, E_train);
4:     COMPUTE-OBJECTIVES(population, s, E_sel);
5:     currentGeneration ← 0;
6:     while currentGeneration ≤ G do
7:         newPop ← ∅ ;
8:         while |newPop| < PS do
9:             p_1 ← SELECTION(population)
10:            p_2 ← SELECTION(population)
11:            if getRandomFloat() < CR then
12:                o_1, o_2 ← crossover(p_1, p_2)
13:            end if
14:            o_1 ← MUTATE(o_1)
15:            o_2 ← MUTATE(o_2)
16:            newPop ← newPop ∪{o_1, o_2}
17:        end while
18:        population ← ELITISM(← newPop ∪ population);
19:        if stagnation detected or currentGeneration ≥ G then
20:            E_sel ← E_sel∪ GET-BEST-INDIVIDUAL(population)
21:        end if
22:    end while
23: end while
24: return E_sel
```

While `INDAGO-Nexus` builds on established MOEAs like `NSGA-II` [57] and `AGE-MOEA` [161], it introduces several domain-specific adaptations: failure-based population seeding, surrogate-based fitness estimation, diversity measured against an evolving archive, and a stagnation-aware reset strategy for enhanced exploration. We describe these customizations in the following paragraphs.

Population Initialization. We seed the initial population with environments where the DRL agent failed during training (E_{train}), biasing the search towards failure-prone regions [24].

Selection. Selection determines which individuals are chosen for reproduction. `INDAGO-Nexus` uses binary tournament selection based on Pareto dominance [57], and the non-dominated sorting in particular.

Objective Calculation. Each individual is evaluated without test execution. The surrogate model s estimates failure likelihood, while diversity is computed with respect to the archive E_{sel} using either Euclidean distance or PCA-based clustering metrics as described in Section 3.3.2.

Crossover and Mutation. `INDAGO-Nexus` applies single-point crossover (Line 12) probabilistically based on the crossover rate CR, randomly selecting a crossover point in the feature vector and swapping segments between parents.

Mutation (Lines 14-15) is *always* applied to both offspring and uses *saliency-based feature selection* [182], which computes the gradient of the surrogate model's output with respect to each input feature. The saliency determines which features impact the surrogate's prediction, guiding mutation decisions. For *fixed-length features* (position, heading), we apply polynomial mutation [55] with type-specific handling (real-valued, integer, binary, categorical). For *variable-length features* (parked vehicle lists), we randomly apply removal, addition, or modification operations. A repair operator ensures validity after mutations.

Elitism. We retain the best individuals using `NSGA-II`'s fast non-dominated sorting with crowding distance [57] or `AGE-MOEA`'s adaptive survival score [161].

Stagnation and Archive Update. `INDAGO-Nexus` maintains an archive E_{sel} of selected scenarios. Stagnation is detected by monitoring the change in hypervolume of the Pareto front P_g at generation g relative to a reference point $\mathbf{r}$ (Equation 3.2). The hypervolume metric reflects both diversity and convergence by measuring the volume dominated by P_g with respect to $\mathbf{r}$:

$$\left|HV(P_g, \mathbf{r}) - HV(P_{g-n_{\text{last}}}, \mathbf{r})\right| < \text{tol} \tag{3.2}$$

When stagnation is detected, `INDAGO-Nexus` selects a representative individual from the Pareto front using two strategies: *Maximum Failure Likelihood* (highest predicted failure probability) or *Knee Point* (best trade-off between failure likelihood and diversity) [234].

3.4. Empirical Evaluation

Our empirical evaluation aims to answer the following research questions:

RQ1 : *How does multi-objective search perform in finding diverse failures?*

RQ2 : *How effective is `INDAGO-Nexus` compared to state-of-the-art `INDAGO`?*

3.4.1. Case Studies

We evaluate `INDAGO-Nexus` on three diverse DRL agents from Biagiola and Tonella [24], each representing different application domains and complexity:

Parking. The first environment we consider is the `parking` environment [129]. In this scenario, the DRL agent begins at a specific location with a defined heading direction. The agent must park in a designated goal lane while avoiding collisions with any of the already parked vehicles. An example of this environment is illustrated in Figure 3.1a. The environment consists of 24 elements, including goal lane, heading, 20 parking slots, and the x, y coordinates of the starting position.

Humanoid. Next, we examine the `Humanoid` environment, one of the more challenging environments from the MuJoCo simulator [206]. In this scenario, the agent

must control a bipedal robot to walk on a smooth surface within a 3D space. The environment configuration of the Humanoid robot consists of two arrays: joint position and joint velocity. The joint position array contains 24 elements representing the positions and rotations of the robot's joints, while the joint velocity array contains the linear and angular velocities of the joints.

Self-Driving Car. The final environment we consider is the Self-Driving Car (SDC), developed using the DonkeyCar simulator [120]. In this environment, the DRL agent must navigate a car from the starting point of a track to the end without leaving track boundaries. The environment configuration consists of a list with 12 pairs, where each pair comprises a command and a value. For example, a command could be a left or right turn, with the value specifying the turn length.

3.4.2. BASELINE

To ensure fair comparison of all approaches, we utilize the original models from the `INDAGO` paper. For our baseline, we employed the GA from the original `INDAGO` tool. This approach uses the predictive value of the surrogate model as a fitness value to gradually mutate environments in the search for failures. We executed this with the `saliency_failure` test policy, which ensures the initial population is created from previously failing environments sourced from the training data, incorporating saliency (as discussed in Section 3.3) during crossover and mutation. We selected this method as it demonstrated the highest rate of failure in the original study [24] against other approaches (`random search` and `hill climbing`). Due to the stochastic nature of evolutionary search, we executed the baseline 50 times, applying the process across all three agent scenarios.

3.4.3. IMPLEMENTATION AND PARAMETER SETTINGS

We implemented `AGE-MOEA` and `NSGA-II` using Pymoo v0.6.1 [32]. Experiments used 50 generations, population size 50, crossover rate 0.75, with stagnation-based termination ($tol = 5 \times 10^{-6}$, $n_{\text{last}} = 10$). The reference point for hypervolume calculation is set to $r = [1.2, 20.2]$, representing the upper bounds for the objectives. Experiments were conducted within Docker containers on a machine equipped with an AMD EPYC 7713 64-Core CPU (2.6 GHz), 256 threads, and an NVIDIA A40 GPU (48GB GDDR6) for tensor-based computation.

3.4.4. EVALUATION CRITERIA

We measure: (1) *unique failures* via PCA+K-means clustering of execution traces, (2) *input/output diversity* using entropy across clusters, and (3) *time-to-failure (TTF)* efficiency. Statistical significance is assessed using Wilcoxon rank-sum test ($\alpha = 0.05$) and Vargha–Delaney effect size.

3.4.5. Failure Detection and Diversity Assessment

To assess the effectiveness of our multi-objective search strategies (**RQ1**), we executed both `NSGA-II` and `AGE-MOEA` across all combinations of diversity metrics and Pareto front selection strategies.

We focused on three aspects: the number of unique failures discovered, the diversity of these failures, and the efficiency in detecting them. We define a failure as a test scenario in which the DRL agent violates its task specification (*e.g.,* crashing, falling, or missing the goal). To measure *unique failures*, we cluster the execution traces (i.e., output trajectories) of failing tests using Principal Component Analysis (PCA) followed by K-means. Each resulting cluster corresponds to a distinct failure type, providing a behavior-grounded notion of uniqueness. Since trajectory lengths vary, we normalize them via zero-padding. For the Parking and SDC scenarios, we track the agent's position over time, while in the Humanoid scenario, we monitor the robot's vertical movement to detect falls.

To determine the optimal number of clusters K^*, we apply silhouette analysis and increase K only if the silhouette score improves by at least 20%, reducing the effect of noisy improvements. This clustering method is used both to compute output diversity and to count unique failures. It is important to distinguish this post hoc clustering from the PCA-based metric used within the search itself (Section 3.3.2). The former relies on test execution results while the latter (our objective) relies on the input features (i.e., without running the tests).

We also evaluate the diversity of failures from both input and output perspectives. *Output diversity* is measured using the same clustering of output trajectories used to identify unique failures. For *input diversity*, we cluster the failing environment configurations using PCA and K-means, providing insight into the structural variety of the input scenarios, regardless of the agent's behavior. In both cases, we calculate two diversity metrics: *unique failures*, which reflects how many clusters are populated, and *entropy*, which quantifies how evenly failures are distributed across clusters. We follow the definitions provided in the `INDAGO` framework [24].

To assess *test efficiency*, we compute the average time required to find the first failing test case. This metric reflects how quickly each approach exposes failures. The same metrics are then used in the comparison between `INDAGO-Nexus` and the baseline `INDAGO` (**RQ2**). To assess statistical significance, we apply the Wilcoxon rank-sum test [49] with a confidence level of $\alpha = 0.05$, and we report the Vargha–Delaney $\hat{A}_{12}$ statistics for the effect size [213].

3.5. Results

This section presents the results of our study, addressing in turn each research question introduced earlier.

Table 3.1. Performance metrics across different configurations for all three agents, displaying median and Interquartile Range (IQR) across 50 runs. (Gray cells indicate top values for each agent and approach.)

(a) Parking Agent

	Failures		Entropy		
Configuration	**#Total**	**#Unique**	**Input**	**Output**	**TTF**
AGE-MOEA					
PCA/Knee	10 (3.75)	7 (1.00)	69.05 (19.19)	62.34 (15.62)	106.12 (26.89)
PCA/O_1	7 (3.00)	4 (2.00)	24.79 (41.62)	39.28 (18.25)	108.13 (26.74)
Euclidean/Knee	10 (3.75)	7 (2.75)	63.95 (18.20)	61.98 (16.33)	123.27 (27.51)
Euclidean/O_1	7 (2.00)	4 (1.00)	11.72 (37.64)	39.85 (14.35)	123.73 (26.27)
NSGA-II					
PCA/Knee	10 (3.00)	7 (2.00)	73.00 (20.13)	63.64 (9.77)	109.19 (23.00)
PCA/O_1	7 (2.75)	4 (1.75)	10.26 (42.01)	41.05 (14.81)	108.12 (25.46)
Euclidean/Knee	11 (3.00)	6 (1.00)	64.12 (14.18)	60.59 (10.26)	120.34 (25.98)
Euclidean/O_1	7 (2.00)	3 (1.00)	0.00 (39.71)	36.12 (15.04)	120.70 (27.56)

(b) Humanoid Agent

	Failures		Entropy		
Configuration	**#Total**	**#Unique**	**Input**	**Output**	**TTF**
AGE-MOEA					
PCA/Knee	-	-	-	-	-
PCA/O_1	2 (1.75)	1 (0.75)	10.65 (34.60)	0.00 (0.00)	78.94 (27.12)
Euclidean/Knee	1 (2.00)	1 (1.00)	0.00 (38.69)	0.00 (0.00)	96.06 (29.95)
Euclidean/O_1	2 (2.00)	1 (1.00)	0.00 (35.59)	0.00 (0.00)	102.53 (27.58)
NSGA-II					
PCA/Knee	-	-	-	-	-
PCA/O_1	2 (1.75)	1 (1.00)	0.00 (32.88)	0.00 (0.00)	79.32 (28.42)
Euclidean/Knee	2 (2.00)	1 (1.00)	0.00 (40.90)	0.00 (0.00)	90.94 (27.65)
Euclidean/O_1	2 (2.00)	1 (0.75)	0.00 (28.95)	0.00 (0.00)	95.70 (23.58)

(c) Self-Driving Car (SDC)Agent

	Failures		Entropy		
Configuration	**#Total**	**#Unique**	**Input**	**Output**	**TTF**
AGE-MOEA					
PCA/Knee	31 (5.75)	5 (6.00)	56.97 (12.51)	63.03 (75.90)	1578.16 (130.69)
PCA/O_1	36 (3.75)	6 (6.00)	71.96 (12.36)	77.34 (86.13)	1092.10 (49.04)
Euclidean/Knee	31 (5.00)	5.5 (6.00)	57.06 (18.83)	71.90 (79.67)	1072.26 (43.36)
Euclidean/O_1	50 (1.00)	6 (6.00)	74.70 (5.84)	64.99 (75.69)	2435.78 (274.89)
NSGA-II					
PCA/Knee	30 (4.00)	5 (6.00)	54.50 (18.05)	62.75 (74.53)	1521.15 (102.29)
PCA/O_1	34 (5.75)	6 (6.75)	70.97 (13.84)	77.00 (85.31)	1087.94 (59.30)
Euclidean/Knee	30 (4.75)	5 (6.00)	56.12 (9.07)	71.07 (80.27)	1073.21 (45.60)
Euclidean/O_1	50 (1.00)	6 (6.00)	72.19 (6.10)	66.78 (74.89)	2588.43 (270.45)

3.5.1. Configuration Comparison (RQ1)

Table 3.1 outlines the performance of each INDAGO-Nexus configuration across the three case studies:

Choice of MOEA: AGE-MOEA generally outperforms NSGA-II in unique failures discovered. In Parking, AGE-MOEA achieves 4-7 unique failures compared to NSGA-II's 3-7. Both perform similarly in SDC, while Humanoid proves challenging for both algorithms. AGE-MOEA's adaptive survival score mechanism appears better suited for the multi-objective DRL testing problem compared to NSGA-II's crowding distance approach.

Diversity Metrics: PCA-based diversity consistently outperforms Euclidean distance across all scenarios. In Parking, PCA configurations achieve substantially higher input entropy (24.79-69.05) compared to Euclidean (11.72-63.95), demonstrating PCA's superior ability to capture meaningful relationships in high-dimensional feature spaces. The Euclidean approach occasionally achieves slightly higher total failure counts, revealing an important trade-off between failure quantity and diversity quality. This suggests that while Euclidean distance may find failures faster in certain regions, PCA-based diversity better explores the full failure space.

Pareto Selection Strategy: Knee-point selection consistently yields superior diversity compared to the extreme O_1 point across all scenarios. The knee point represents a balanced trade-off between objectives, achieving higher unique failures and entropy scores.

> Different configurations perform best in different scenarios. Overall, AGE-MOEA tends to produce more unique and diverse failures than NSGA-II. Pareto knee-point selection consistently yields better results than the extreme O_1 point. However, the choice of configuration can significantly impact performance, highlighting the need for adaptive strategies that consider the specific context and objectives.

3.5.2. Comparison with INDAGO (RQ2)

Table 3.2 presents a direct comparison between INDAGO-Nexus (AGE-MOEA *Euclidean/Knee* configuration) and the state-of-the-art INDAGO approach. The results demonstrate significant improvements across multiple metrics:

Unique Failure Discovery: INDAGO-Nexus consistently finds more unique failures than INDAGO. In the Parking scenario, INDAGO-Nexus discovers 7 unique failures compared to INDAGO's 5 (+40% improvement). For the SDC scenario, INDAGO-Nexus finds 5.5 unique failures versus INDAGO's 3 (+83% improvement). The Humanoid scenario shows comparable performance between both approaches, which aligns with our observation that this scenario is inherently challenging for all testing methods.

Table 3.2. Comparison between INDAGO-Nexus and INDAGO across different agents, displaying median and Interquartile Range (IQR) across 50 runs. Statistical significance markers for the Wilcoxon rank-sum test: * p-value<0.05, ** p-value<0.01, *** p-value<0.001.

(a) Parking Agent

	Failures		Entropy		
Approach	**#Total**	**#Unique**	**Input**	**Output**	**TTF**
INDAGO-Nexus	10 (3.75)	7 (2.75)**	63.95 (18.20)***	61.98 (16.33)***	123.27 (27.51)***
INDAGO	14.5 (4.00)	5 (2.00)	0.00 (35.72)	49.37 (12.56)	150.53 (0.03)

(b) Humanoid Agent

	Failures		Entropy		
Approach	**#Total**	**#Unique**	**Input**	**Output**	**TTF**
INDAGO-Nexus	1 (2.00)	1 (1.00)	0.00 (38.69)	0.00 (0.00)	96.06 (29.95)***
INDAGO	2 (2.75)	1 (1.00)	28.90 (39.31)	0.00 (50.89)	150.50 (0.01)

(c) Self-Driving Car (SDC) Agent

	Failures		Entropy		
Approach	**#Total**	**#Unique**	**Input**	**Output**	**TTF**
INDAGO-Nexus	31 (5.00)***	5.50 (6.00)**	57.06 (18.83)	71.90 (79.67)***	1072.26 (43.36)***
INDAGO	17.00 (5.75)	3.00 (3.00)	61.72 (21.59)	34.36 (48.71)	3251.97 (0.35)

Diversity Enhancement: The multi-objective approach significantly improves both input and output diversity. In the Parking case, INDAGO-Nexus achieves 63.95 input entropy compared to INDAGO's 0.00, and 61.98 output entropy versus 49.37. This demonstrates that explicitly optimizing for diversity as an objective leads to more varied test scenarios and failure modes. The diversity improvements are particularly important for comprehensive testing, as they increase the likelihood of discovering edge cases and systematic weaknesses.

Efficiency Gains: INDAGO-Nexus demonstrates superior efficiency in finding failures across all scenarios. Time-to-failure improvements range from 18% (Parking: 123.27 vs 150.53) to 67% (SDC: 1072.26 vs 3251.97), with Humanoid showing 36% improvement (96.06 vs 150.50). These efficiency gains suggest that the multi-objective approach not only finds more diverse failures but does so more quickly than single-objective methods. The efficiency improvements are statistically significant (Wilcoxon test, $p < 0.001$) with large effect sizes across all scenarios (Vargha-Delaney $\hat{A}_{12} > 0.9$) for TTF.

Statistical Significance: All reported improvements show statistical significance at $\alpha = 0.05$ level using the Wilcoxon rank-sum test. Effect sizes calculated using Vargha-Delaney $\hat{A}_{12}$ statistics indicate substantial practical differences, particularly for efficiency metrics where INDAGO-Nexus consistently achieves large effect sizes ($\hat{A}_{12} > 0.7$), indicating that INDAGO-Nexus significantly outperforms INDAGO in key performance areas.

INDAGO-Nexus finds more unique failures and higher input/output diversity in Parking and SDC. Instead, it matches INDAGO in the challenging Humanoid scenario. Finally, INDAGO-Nexus finds unique failures faster across all DRL agents.

3.6. Threats to Validity

Internal Validity: Our experimental design addresses several potential threats through careful control of variables and statistical rigor. We ensure fair comparison by running all algorithms under identical conditions with the same hardware and time constraints. Each experiment is repeated 50 times to account for the stochastic nature of evolutionary algorithms, and we apply appropriate statistical tests (Wilcoxon rank-sum) with effect size measurements (Vargha-Delaney $\hat{A}_{12}$) to assess practical significance. The clustering approach for measuring unique failures uses established silhouette analysis with conservative thresholds (20% improvement requirement) to avoid over-segmentation due to noise.

External Validity: Our evaluation covers three diverse DRL domains with different state/action spaces and failure modes. However, generalizability to other DRL applications, training algorithms beyond PPO, or different architectures requires further investigation.

Construct Validity: The choice of evaluation metrics reflects established practices in the DRL testing literature, particularly building on the INDAGO framework. Our clustering-based approach to measuring unique failures provides a behavior-grounded assessment of diversity that captures meaningful differences in agent behavior rather than superficial input variations. The entropy-based diversity measures provide quantitative assessments that complement the qualitative notion of test case variety.

Conclusion Validity: The statistical methods employed (non-parametric tests, effect size calculations) are appropriate for the experimental design and address the non-normal distribution of performance metrics commonly observed in evolutionary computation experiments. The large number of experimental repetitions (50 per configuration) provides sufficient statistical power to detect meaningful differences between approaches. Furthermore, a small number of SDC configurations were executed on a smaller hardware setup (AMD Ryzen Threadripper 3970X 32-Core Processor, 64GB memory, NVIDIA GeForce RTX 3080 10GB). Although less powerful then the main setup, the results were considerably faster. However, other validation metrics remain consistent.

3.7. Conclusion and Future Work

This paper introduced INDAGO-Nexus, a multi-objective search approach for generating diverse failures in DRL environments. Unlike prior methods that focus solely on inducing agent failures, INDAGO-Nexus simultaneously optimizes for failure likelihood and scenario diversity, leading to a more comprehensive exploration of failure

modes. Our empirical evaluation across three DRL environments—Parking, Humanoid, and Self-Driving Car (SDC)—shows that the multi-objective `INDAGO-Nexus` consistently outperforms the single-objective baseline `INDAGO` in terms of unique failures discovered, diversity (measured via input and output entropy), and search efficiency.

We found that the effectiveness of `INDAGO-Nexus` varies across scenarios, with `AGE-MOEA` generally achieving more unique and diverse failures than `NSGA-II`. Both the choice of diversity metric (Euclidean vs. PCA) and the Pareto selection strategy (knee point vs. extreme O_1) significantly influence the outcomes. In particular, knee-point selection consistently led to better results across most configurations. Nonetheless, the Humanoid scenario remains particularly challenging, exposing limitations in current diversity metrics and search robustness.

Future Work: Several research directions emerge from this work. First, exploring many-objective optimization with additional objectives such as test case complexity, execution cost, or coverage metrics could further improve testing effectiveness. Second, investigating adaptive diversity metrics that automatically adjust based on the characteristics of discovered failures could enhance the approach's generalizability. Third, extending the framework to other DRL training algorithms (beyond PPO) and neural network architectures would broaden its applicability. Finally, developing theoretical foundations for understanding when and why multi-objective approaches outperform single-objective methods in testing contexts would provide valuable insights for the testing community.

`INDAGO-Nexus` introduces multi-objective search for generating diverse DRL failures, consistently outperforming `INDAGO` in unique failures and efficiency. `AGE-MOEA` with knee-point selection proves most effective, though Humanoid scenarios remain challenging. Future work includes many-objective extensions, improved surrogate models, and broader domain evaluation.

4

SNAKES ON A 2-D PLANE: GEOMETRY-BASED IMAGE REPRESENTATIONS FOR TEST CASE SELECTION IN AUTONOMOUS DRIVING SIMULATION

Test case selection is critical for efficient testing of self-driving cars (SDCs), where executing all generated test cases in simulation is prohibitively expensive. Existing SDC test selection techniques rely on sequential or engineered feature representations, while recent Convolutional Neural Network (CNN)-based image approaches explore only limited geometric encodings and architectures. In this work, we conduct a thorough empirical evaluation of geometry-based image representations for SDC test case selection. We curate three distinct image datasets, namely simple road plot, geometric heatmap, and geometric points, encoding features including jerk, chord deviation, and path angle changes. Through road pair analysis of approximately 10,000 PASS and FAIL cases, we find that abrupt geometric changes are more predictive of failure than absolute curvature values. Guided by these findings, we optimize six CNN architectures, spanning custom designs, fine-tuned VGG models, and two architectures adapted from SnakeCLEF competition submissions. The latter are motivated by the visual appearance of the geometric heatmap representation of the roads—a narrow, banding-colored, continuously-varying curve on a plain background—which shares a striking resemblance with snake imagery. We identify optimal hyperparameters using a two-stage optimization process and

implement an adaptive diversity-aware selection algorithm that adjusts test selection thresholds based on predicted failure probabilities. We evaluate our approach on a newly created dataset containing approximately 16,000 test cases and compare it against state-of-the-art SDC test selection techniques, including `ITS4SDC`, `RoadGeom` and `DRVN`. We find that CNN-based approaches vary in performance, with our EfficientNet-based model achieving the best time-to-fault and fault-to-selection ratios, while custom geometric CNNs obtain the highest diversity.

This chapter is currently under review in the Journal of Systems Software (JSS), by Antony Bartlett, Cynthia C. S. Liem and Annibale Panichella. See Chapter 1.6 for additional publication details.

4.1. Introduction

Self-driving cars (SDCs) are increasingly deployed in safety-critical contexts, where failures can have severe consequences for both passengers and surrounding traffic participants [164]. As a result, the correctness and reliability of autonomous driving systems are of paramount importance. To address these concerns, simulation-based testing has become a central component of the development and validation process for SDCs [27]. Modern simulation environments enable developers to explore a wide range of driving scenarios under controlled conditions, allowing for safe, repeatable, and cost-effective testing compared to real-world experimentation [65,134,179]. However, despite these advantages, simulation-based testing introduces its own challenges.

One fundamental challenge lies in the sheer expanse of the scenario space [20]. Even when restricting attention to road geometry and vehicle dynamics alone, the number of possible test cases grows combinatorially, quickly exceeding what can be executed within realistic time and resource constraints. Prior work has shown that executing all generated simulation tests during testing is often infeasible, as many tests contribute little to fault detection while still incurring substantial execution costs [25]. Consequently, effective testing for SDCs requires techniques that can prioritize or select test cases that are more likely to expose failures early.

Test case selection (TCS) and prioritization (TCP) have been widely studied in traditional software engineering [229], where historical execution data is often used to guide selection decisions [149]. However, this has been little explored in the context of SDCs, where historical data can contain complex information such as road geometry, vehicle parameters, and environmental constraints. This creates a new challenge, in that methods must now extrapolate this information to understand the failure points in the data.

Recent research has explored a variety of approaches to address this challenge, such as SDC test selection techniques that rely on sequential models or engineered geometric features derived from road descriptions using machine learning models such

as Long Short-Term Memory (LSTM), or transformer-based architectures to estimate failure likelihood [9,25,82,96]. These approaches have demonstrated strong performance in recent tool competitions [28,29], but they typically operate on abstracted representations of road geometry and do not explicitly capture the spatial structure inherent in road layouts. In contrast to these methods, we attempted to apply Convolutional Neural Networks (CNNs) to the problem of SDC test case selection.

In this paper, we extend upon our previous work—presented at the Internation Conference on Software Testing (ICST)'25 competition [17,28]—which introduced the idea of using CNNs for test case selection in SDCs. By transforming road geometry information into simple plotted images, we applied a fine-tuned VGG19 [183] model to guide our greedy search algorithm. We now extend that work, as we seek to understand which geometric properties of road layouts are most indicative of failure. By performing a systematic empirical study of geometry-based image representations, we can better understand how data should be encoded to support image-based selection for SDC TCS. We analyze large collections of simulated road scenarios and perform a detailed road-pair analysis of matched PASS and FAIL cases, showing that abrupt geometric changes, such as those captured by jerk, chord deviation, and path angle variation, are more predictive of failures than absolute curvature values alone.

Guided by these findings, we design and evaluate multiple image representations of road geometry and train a diverse set of CNN architectures for TCS. Two custom architectures are designed to extract features from multi-channel geometric heatmaps, while fine-tuned versions of VGG16 and VGG19 [183] are used to analyze simpler road plot images. Lastly, two of the six architectures are sourced from SnakeCLEF competition submissions [167,168], motivated by the visual appearance of the geometric heatmap road visual, whose banding-colored, continuously-varying curves bear a specific resemblance to snake imagery, rather than by a general analogy between roads and snakes. Beyond failure prediction, we also address the challenge of selecting diverse test suites by incorporating an adaptive diversity-aware selection strategy that balances predicted failure likelihood with coverage of distinct scenarios.

We evaluate our approach on a newly curated dataset of approximately 16,000 simulated road scenarios and compare it against state-of-the-art SDC test selection techniques; `ITS4SDC` [82], and `RoadGeom` [9], together with our original CNN approach, `DRVN` [17]. Our results show that geometry-based CNN models achieve competitive fault detection efficiency compared to existing sequential and feature-based approaches on key effectiveness metrics. In particular, our best-performing model achieves competitive time-to-fault ratio and fault-to-selection ratio among all evaluated techniques, while geometry-aware CNNs consistently obtain higher test suite diversity. These results indicate that image-based representations of road geometry can effectively support TCS for SDCs when combined with structure-aware models and adaptive diversity handling.

Therefore, our investigation was guided by the following research questions (RQs):

RQ1 : *To what extent does dataset choice affect performance of the Convolutional Neural Network (CNN) architectures?*

RQ2 : *Which CNN architectures perform best for test case selection when compared to existing state-of-the-art techniques?*

Our contributions can be summarized as follows:

- We conduct a large-scale empirical study of road geometry in SDC simulations, showing that abrupt geometric changes are more predictive of failures than curvature.
- We design three geometry-based image representations of road layouts and evaluate them across six CNN architectures, including models inspired by snake-identification tasks, to study geometry-aware test case selection.
- We propose an adaptive diversity-aware test selection strategy that balances predicted failure likelihood with test suite diversity.
- We empirically compare our approach against state-of-the-art SDC test selection techniques on a large unseen dataset of simulated road scenarios.
- We provide a full replication package via **Figshare**[1] and a new dataset of approximately 16,000 simulated road scenarios to support future research in SDC test case selection.

4.2. Background and Related Work

4.2.1. Test Case Selection for SDC Simulation Testing

Test case selection (TCS) and test case prioritization (TCP) have been widely studied in traditional software engineering as techniques to reduce overall test execution cost by identifying tests most likely to reveal defects [149,229]. In classical settings, TCS operates in a regression context where two system versions exist and tests are selected based on which parts of the code have changed; TCP reorders a fixed test suite to detect failures as early as possible. Both paradigms commonly exploit historical execution data, such as previous test results or code coverage metrics, to guide selection decisions.

In the context of self-driving cars (SDCs) however, neither of these assumptions holds directly. There is typically a single, fixed system under test; test cases are generated synthetically and selected before execution, leaving no prior outcome history to exploit. Consequently, TCS for SDCs is better characterised as a failure-prediction problem: given only the static description of a new test scenario (*e.g.,* road geometry), estimate its likelihood of revealing a failure and select accordingly.

[1] https://doi.org/10.6084/m9.figshare.30906746

This problem therefore operates in a black-box setting [26], without access to system internals, change history, or previous execution outcomes.

4.2.2. Simulation-based testing for self-driving cars

Simulation-based testing is widely used for validating autonomous driving systems because it enables systematic exploration of driving scenarios that would be unsafe, expensive, or impractical to execute in the real world [65,134,179]. In recent years, researchers have highlighted broader challenges associated with simulation-based testing of autonomous cyber-physical systems (CPS), including the need for improved simulation quality metrics, scenario generation methods, and methods to assess the effectiveness of testing strategies [27].

Within CPS and autonomous driving contexts, simulation based testing is typically embedded throughout system development, including model-in-the-loop, software-in-the-loop, and hardware-in-the-loop stages. Although such staged simulations facilitate safe and repeatable evaluations, they also introduce scalability challenges further amplified in autonomous driving, as the number of generated scenarios grows, the execution and management of these tests becomes increasingly costly and time-consuming [27]. This has motivated research in automated test generation and selection techniques that can reduce execution cost while preserving fault detection capability, especially within the continuous validation workflows typical of DevOps-style SDC development [25,27].

As a result, there is a growing interest in techniques that go beyond test generation and explicitly support TCS and TCP for simulation-based SDC testing. In the context of SDCs, such techniques must operate on static scenario descriptions and scale to large numbers of generated road configurations, motivating learning-based approaches that can infer failure likelihood from scenario characteristics.

4.2.3. Learning-based test case selection for SDCs

Recent work has explored learning-based techniques for TCS in the context of simulation-based tests of SDCs. A comprehensive study by Birchler *et al.* [25] introduced the SDC-Scissor tool, which extracts geometric and statistical features from road layouts to classify test cases for lane-keeping systems. Their work demonstrated that machine learning models can effectively predict failure-prone scenarios based on engineered road features.

Building on this line of research, a variety of approaches have been proposed as part of the ICST and SBFT SDC testing competitions [28,29]. Our approach for this competition (`DRVN`) [17] represents a novel CNN-based method, transforming road geometries into simple plotted images, using a single architecture for feature extraction. While effective, it is limited by its simplistic visual encoding and lack of architectural exploration.

Some state-of-the-art techniques adopt sequential representations of road geometry. The SBFT 2025 competition winner, `RoadGeom` [9], employs a transformer encoder to capture long-range dependencies in road sequences using self-attention mechanisms. Similarly, the ICST 2025 competition winner, `ITS4SDC` [82], utilizes a Long Short-Term Memory (LSTM) network to model sequential road data and predict failure likelihood. These approaches achieve strong performance but operate on abstracted representations of road geometry and do not explicitly exploit its spatial or visual structure.

In contrast, our work focuses on understanding how road geometry should be represented to support learning-based TCS. We systematically explore geometry-aware image representations and CNN architectures for TCS in SDCs, studying how different spatial encodings and inductive biases affect failure prediction and test selection effectiveness.

4.3. Approach

In this section we discuss the implementation of our approach. We first discuss the creation of our model training data, followed by the models we trained. Next we discuss the approach we took to optimize the hyperparameters of our models and finally the method with which we select tests during the final evaluation.

4.3.1. Training data generation

4.3.1.1. Road data

The training data for our CNN models is a curated set of road scenarios, originally created by Birchler *et al.* [31] using the SDC-Scissor tool [25]. This dataset contains a diverse set of road geometries, each associated with a PASS or FAIL outcome based on actual execution in the BeamNG simulation environment [19]. The dataset contains over 30,000 test cases, with around a 60:40 split between PASS and FAIL outcomes. A test case is labeled as FAIL when the vehicle departs the lane beyond a predefined out-of-bound (OOB) threshold during simulation [25].

A single test case consists of metadata outlining the conditions of the test, such as the maximum vehicle speed, the maximum allowed OOB, and road data. The road data consists of sequences of 2D coordinate points (x, y) that define the geometry of each road scenario. Each test case contains up to 200 road points, where each point connects sequentially to form a continuous path representing the road centerline.

To generate our specific training data, we consider a number of geometric features of this dataset—as discussed in literature [159,169]—that can be extracted from the road points. We perform the following analysis of these features:

(a) Curvature: Computed as the rate of change of the road's tangent direction, calculated using the second derivative of the coordinate sequence. High curvature values indicate sharp turns that may challenge autonomous vehicle control systems.

(b) Jerk: The third derivative of position, representing the rate of change of acceleration. High jerk values indicate abrupt changes in curvature, which can cause instability in vehicle dynamics and control algorithms.

(c) Chord Deviation: Measures how much each road point deviates from the straight-line chord connecting nearby points. This captures local variations in road geometry that may not be evident from curvature alone.

(d) Path Angle Change: The difference in heading angle between consecutive road segments, normalized to $[-\pi, \pi]$. Large angle changes indicate sharp directional changes that can be problematic for path planning algorithms.

(e) Curvature Gradient: The rate of change of curvature along the road, computed as the first derivative of the curvature sequence. This captures how quickly the road geometry transitions between different curvature regions.

These geometric features are computed for each point along the road, using NumPy's[2] gradient operator, which employs central differences that provide implicit local smoothing compared to simple forward and backward differences. This approach accentuates high-frequency geometric variations that may indicate failure conditions. The chord deviation feature incorporates a local window of 5 points that captures deviation from local linear approximations, while boundary effects are handled by the gradient operator's built-in edge handling. This approach prioritizes geometric fidelity and the preservation of potentially critical failure indicators, for distinguishing passing and failing road geometries.

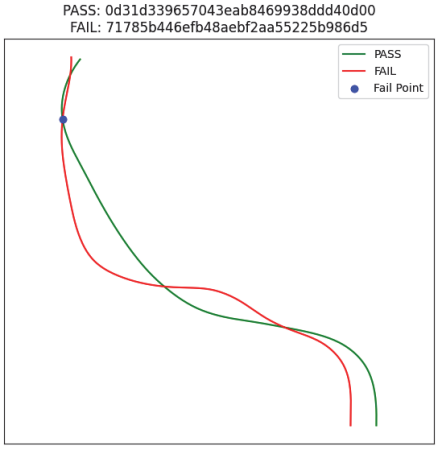


Figure 4.1. Estimation of failure point along a road using time to failure.

[2]https://numpy.org/

4.3.1.2. ROAD PAIR ANALYSIS

To identify which geometric features have the strongest discriminative power between passing and failing executions, we developed a systematic approach to analyze similar road pairs with different outcomes:

(a) Semantic Road Matching: We implemented a geometry-based similarity matching method (Algorithm 4.1) to identify pairs of roads with similar geometric characteristics but different test outcomes.

Roads in the dataset vary substantially in length, a variation reflecting differences in road complexity and curvature density. However, two roads can have nearly identical geometric shapes while differing in total length, *e.g.,* one road may follow a meandering path compressed into fewer segments, while another follows the same topological pattern but over more distance. To enable fair comparison of such geometrically similar but length-varying roads, we resample each road to a uniform number of points ($n = 100$) using arc-length parameterization [217]. This approach preserves the geometric shape (ensuring that tight curves remain tight and gentle curves remain gentle) while enabling direct point-to-point comparison. The choice of $n = 100$ balances fidelity (sufficient resolution to capture local geometric variation) with computational efficiency and numerical stability—smaller values ($n < 50$) risk oversimplifying curved sections, while larger values ($n > 200$) introduce unnecessary computational overhead without materially improving matching accuracy.

(b) Failure Point Estimation: For each FAIL road paired with a similar PASS road, we estimate the likely failure point using temporal information from the test execution (Figure 4.1). Given that tests include a "time to failure" metric, we interpolate along the road's length to identify the approximate (x, y) coordinate where the failure occurred, assuming roughly constant vehicle speed:

$$(x_f, y_f) = \text{interpolate}\left(\text{road_points}, \frac{t_{fail}}{t_{total}}\right) \tag{4.1}$$

where t_{fail} is the time to failure and t_{total} is the estimated total traversal time. This approach assumes a constant vehicle speed along the road, which should be relatively consistent due to the similartiy of road shapes compared in the matching process.

(c) Statistical Feature Analysis: To rigorously identify which geometric features discriminate between PASS and FAIL outcomes, we applied Mann-Whitney U tests (non-parametric comparison of two independent groups) and computed effect sizes using rank-biserial correlation (r) and Area Under the Receiving Operator Characteristic curve (AUC). We also computed Spearman's rank correlation coefficient (ρ) [173], a non-parametric measure of monotonic association between two variables based on ranked values. This allows us to assess whether features are correlated (redundant) or independent (complementary). For each feature f and failure point i, we computed statistical measures including mean, standard deviation, maximum, and z-score within a window of ± 10 points around the failure location.

Algorithm 4.1 Similarity-based Road Matching

Require: R_{pass} = set of PASS roads, R_{fail} = set of FAIL roads, n = resample size
Ensure: P = set of matched road pairs (PASS, FAIL)

$P \leftarrow \emptyset$
for each $r_p \in R_{pass}$ **do**
  $pts_p \leftarrow$ resample_arc_length($r_p.points$, n)
  $pts_p \leftarrow$ normalize(pts_p) ▷ Center and scale
  $t_p \leftarrow r_p.duration$
  $best_match \leftarrow$ null, $best_dist \leftarrow \infty$
  for each $r_f \in R_{fail}$ **do**
    $t_f \leftarrow r_f.duration$
    if $t_p > t_f$ **then** ▷ PASS must outlast FAIL
      $pts_f \leftarrow$ resample_arc_length($r_f.points$, n)
      $pts_f \leftarrow$ normalize(pts_f)
      $dist \leftarrow \text{mean}(||pts_p - pts_f||_2)$ ▷ Euclidean distance
      if $dist < best_dist$ **then**
        $best_match \leftarrow r_f$, $best_dist \leftarrow dist$
      end if
    end if
  end for
  if $best_match \neq$ null **then**
    $P \leftarrow P \cup \{(r_p, best_match, best_dist)\}$
  end if
end for
return P

Table 4.1. Statistical justification for feature selection. Mann-Whitney U test results comparing PASS and FAIL roads across geometric features. Effect size r and AUC quantify discriminative power; p-values test significance of the difference.

Feature	PASS μ	FAIL μ	Δ%	p-value	Effect r	AUC
Curvature	0.0277	0.0275	–0.9%	7.9E-3	0.032	0.516
Jerk	**0.0041**	**0.0048**	**+15.4%**	**2.2E-74**	**0.218**	**0.609**
Curvature Gradient	0.0001	0.0001	+3.8%	1.6E-3	0.038	0.519
Chord Deviation	**0.2628**	**0.3016**	**+14.8%**	**1.4E-59**	**0.195**	**0.597**
Path Angle Change	**0.0014**	**0.0002**	**–88.9%**	**1.7E-5**	**0.051**	**0.526**

Our analysis of ≈10,000 matched road pairs (spanning 3,101 FAIL and 9,343 PASS cases) applied Mann-Whitney U testing [137] to evaluate the discriminative power of five candidate geometric features. Results are summarized in Table 4.1 where we observe three features demonstrating statistically significant discriminative power (all $p < 0.001$) with meaningful effect sizes and AUC scores substantially above

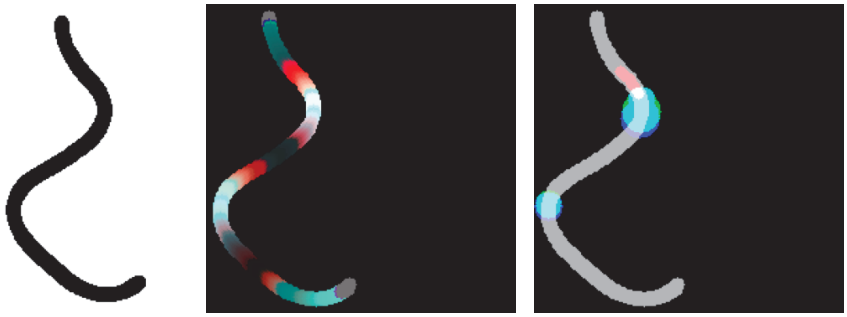
0.5.

Feature Selection Justification: The three selected features (jerk, chord deviation, path angle change) demonstrate superior statistical properties compared to curvature and curvature gradient:

- **Jerk**: Highest AUC (0.609) and effect size ($r = 0.218$) with highly significant p-value (2.2×10^{-74}). This indicates that abrupt changes in curvature are the strongest individual predictor of failure.
- **Chord Deviation**: Substantial discriminative power (AUC = 0.597, $r = 0.195$, $p = 1.4 \times 10^{-59}$), capturing local geometric irregularities.
- **Path Angle Change**: While showing lower effect magnitude, it provides independent information not captured by jerk or chord deviation (Spearman $\rho \approx 0.04$ with other features), covering directional heading changes.
- **Redundancy Analysis**: Curvature and curvature gradient are highly correlated (Spearman $\rho = 0.284$) and show weak discriminative power, making them redundant; the three selected features maintain low mutual correlation (mean$|\rho| = 0.33$), ensuring complementary information.
- **Mutual Information**: Jerk ranks highest (MI = 0.0183 nats), followed by chord deviation (0.0170 nats), and path angle change (0.0070 nats), confirming their relative utility for predicting the PASS/FAIL label.

4.3.1.3. DATASET GENERATION

The central aim of this work is to understand how CNNs can be applied to SDC TCS. CNNs are inherently designed to operate on two-dimensional spatial inputs [128]: they learn hierarchical features through convolutional filters applied over a fixed-size image grid (*e.g.,* 224×224 pixels), and the broad availability of pretrained models further reinforces this standard interface. Given the ability to visualize a road as a 2D layout, we can leverage CNNs to learn spatial patterns in the road geometry that may correlate with the failure likelihood of a test case. While a road is defined as a 1D sequence of points, this sequence traces a path through 2D coordinate space, and projecting it onto a 2D image plane directly preserves the spatial layout of that path. This approach is consistent with a wider pattern in the literature: Hatami *et al.* [90] show that transforming 1D time-series into 2D image representations enables CNNs to exploit feature types that are not available from raw sequential signals. In the autonomous driving domain, Cui *et al.* [53] similarly rasterize scene context—including road geometry and agent trajectories—as 2D images and apply deep CNNs for trajectory prediction, demonstrating the effectiveness of image-based spatial encodings for path-structured data. In contrast to sequential models that operate on the raw coordinate stream, the image representation allows CNN architectures to exploit learned spatial filters and established pretraining, without requiring custom sequence-handling designs.



(a) (b) (c)

Figure 4.2. Example training dataset images: (a) Simple road plot, (b) Geometric heatmap, (c) Geometric points.

To facilitate the use of CNNs for TCS, we must first transform the test cases into a format suitable for image-based analysis. Each test case consists of a sequence of road points that define the geometry of a road scenario. Our original approach (`DRVN`), used `Matplotlib`[3] to plot these points as a simple line, creating a visual representation of the road geometry. However, Matplotlib is primarily designed for visualization rather than large-scale data generation, which can introduce unnecessary overhead when constructing large image datasets or supporting repeated inference. Therefore, we explore the use of `OpenCV` (cv2)[4] as an alternative for image generation.

Road coordinate data is defined on a 200×200 plane and are linearly scaled to generate 224×224 images—a standard input size for many CNN architectures [92,183]. This choice allows direct use of pre-trained models while preserving the spatial structure of road geometries. For further consistency, each dataset uses images with three channels (RGB), encoding multiple geometric features simultaneously. Furthermore, to understand the feature extraction of our models, each dataset is created as two variants with an arbitrarily chosen line thickness of 5 and 9 pixels.

4.3.1.4. SIMPLE ROAD PLOT

Similar to our existing work [17], we plot a raw example of the road (Figure 4.2a). This is the simplest representation of the road points, where each point connects by a line to form a continuous road. This captures basic geometry of the road, including curves and straight segments.

[3] https://matplotlib.org/
[4] https://opencv.org/

4.3.1.5. GEOMETRIC DATA

We generate two geometry-aware image representations encoding road features either continuously along the road trajectory or sparsely at salient locations. In both cases, images encode multiple geometric properties using the RGB channels: `Red channel` represents jerk magnitude, the `Green channel` encodes chord deviation, and the `Blue channel` captures path angle changes. Feature values are normalized to ensure consistent intensity scaling across datasets.

Geometric Heatmap: The geometric heatmap (Figure 4.2b) encodes feature magnitudes, visualized by coloring segments along the road trajectory. To reduce boundary artifacts introduced by derivative-based features, values near the start and end of the road are replaced with the median feature value. The resulting image is a narrow, elongated curve whose colour varies continuously along its length against a plain background. This specific visual structure—distinct from a plain road plot—bears a similar banding-color imagery encountered in snakes, heavily studied in the SnakeCLEF challenge [167,168], and it directly motivates our evaluation of two CNN architectures originating from that competition.

Geometric Points: The point-based dataset generation process focuses on highlighting critical geometric features at specific road points (Figure 4.2c). This method emphasizes the top 90th percentile of feature values, drawing attention to the most significant points along the road. The road is visualized as a gray line, giving context to the highlighted features. To avoid boundary artifacts, feature values near the start and end of the road are excluded from the top percentile calculations.

Finally, for all datasets, we store the image as a raw byte array, together with the associated label (PASS/FAIL) in a TFRecord file.

4.3.2. CONVOLUTIONAL NEURAL NETWORK ARCHITECTURES

We evaluate six Convolutional Neural Network (CNN) architectures; two custom networks (GF-CNN and RES-CNN) designed specifically for our geometric image representations, two fine-tuned VGG models (VGG16 and VGG19), following our existing `DRVN` tool setup [17], and two more complex architectures (EFNB4 and CNV2MSC) adapted from top-performing models in the SnakeCLEF competition [167,168]. As discussed in Section 4.3.1.3, the choice of the EFNB4 and CNV2MSC architectures is motivated by the visual appearance of the geometric heatmap representation, whose banding-colors and continuously-varying curves, resemble the elongated structures in snake imagery. Beyond this visual motivation, both architectures bring strong inductive biases independently well-suited to our task: EfficientNet-B4 [203] uses compound scaling to balance depth, width, and resolution efficiently, while the ConvNeXtV2-style network employs depthwise convolutions and explicit multi-scale feature fusion.

4.3.2.1. pretrained backbone: efficientnet-b4 (fine-tuned)

Based on a model from the SnakeCLEF 2022 competition [168], we adapt the work of Bloch *et al.* [33] who utilized EfficientNet-B4 [203] for snake species identification. This architecture is well-suited for our task due to its balance of accuracy and computational efficiency. We initialized the model with weights pretrained on ImageNet [58], removing the classification head and fine-tuning only later convolutional blocks (Blocks 5–7), while freezing all batch normalization layers to preserve pretrained statistics. The classification head comprises global average pooling, a dense projection with parameterized activation, layer normalization, dropout and a margin-based softmax head (ArcFace [59]) that produces logits over our two classes (PASS/FAIL).

4.3.2.2. geometric feature cnn

This custom architecture is a more standard CNN design, containing a stack of four convolutional blocks (32, 64, 128, 256 channels), each comprising convolution, batch normalization, ReLU activation, and max pooling, with progressively increased dropout. An optional 1×1 attention map gates the final feature map. As above, we concatenate global average and max pooled features, followed by two dense layers with dropout and L2 regularization, and produce binary logits trained with sparse categorical cross-entropy from logits. This model provides a strong, parameter-efficient baseline tailored to three-channel geometric inputs.

4.3.2.3. convnextv2 multi-scale cnn

We designed a ConvNeXtV2-style [224] network with a compact backbone and an explicit multi-scale fusion pathway. Inspired by the SnakeCLEF 2023 competition models from Hu *et al.* [101] and Bracke *et al.* [35], we adapt a ConvNeXtV2 architecture. The backbone stacks ConvNeXt blocks composed of depthwise convolutions (implemented via grouped convolutions), layer normalization, pointwise expansions, GELU activations, and lightweight squeeze-and-excitation refinements. We construct a feature pyramid network (FPN) over four hierarchical stages and apply a CBAM attention module to jointly recalibrate channels and spatial locations. The classification head mirrors the pretrained model's head with global average pooling, a dense projection, dropout, and an ArcFace margin head for binary logits.

4.3.2.4. residual geometric cnn

This custom architecture adopts a ResNet-inspired [92] design with three residual stages (32, 64, 128 channels). Dimension changes are handled with 1×1 projection shortcuts. An attention mask (1×1 sigmoid) is applied after the final stage to emphasize salient spatial regions. We combine complementary global descriptors via average and max pooling and pass the concatenated representation through two

dense layers with dropout and L2 regularization. The network outputs logits over two classes and is trained with sparse categorical cross-entropy from logits.

4.3.2.5. VGG16 and VGG19 fine-tuned CNN

We adapt the code from the DRVN [17] approach, where we fine-tuned a VGG19 [183] architecture to create the model. We extend this by also fine-tuning a VGG16 [183] architecture for comparison. Both models are initialized with weights pretrained on ImageNet, with the final four layers unfrozen for fine-tuning. The classification head consists of two dense layers with dropout, followed by a final dense layer with sigmoid activation to output binary predictions.

4.3.3. Parameter optimization

In this section, we outline our approach to hyperparameter optimization for the various CNN architectures evaluated in this study. To identify optimal hyperparameters for each CNN architecture, we employ a two-stage optimization process—Random Search and Stochastic Hill-Climbing—where the best performing model based on Macro-F1 score [94] is selected.

4.3.3.1. Random search

Initially, we conduct a random search (RS) over a predefined hyperparameter space, to explore a wide range of configurations. This method was deemed preferable to grid search, due to the high dimensionality of the hyperparameter space and the computation cost, as discussed by Bergstra and Bengio [21]. Given the scope of parameters, even for our smaller models, grid search would require ≈5103 combinations. If we consider an average training time of 20 minutes per model, this would result in an estimated total training time of approximately 71 days. Even considering parallelization, this would still be infeasible. Therefore, we opt for RS, which allows us to sample a diverse set of hyperparameter combinations more efficiently. The hyperparameters for our search are outlined in Table 4.2.

We randomly sample 200 hyperparameter combinations for each architecture and dataset combination, training each model for 20 epochs. The performance of each model is evaluated using the Macro-F1 score on a validation set. The top scoring architecture/ dataset combination from this stage is selected for further refinement.

4.3.3.2. Stochastic hill-climber

Following the RS, we employ a stochastic hill-climbing (SHC) algorithm [111] to refine the hyperparameters of the best-performing of each model architecture from the previous stage. This method iteratively explores the hyperparameter space by

Table 4.2. Hyperparameter search space.

Hyperparameter	Values
Learning Rate	[1e-4, 5e-4, 1e-3]
Batch Size	[16, 32, 64]
Dense Units	[128, 256, 512]
Dropout Rate	[0.3, 0.5, 0.7]
L2 Regularization Strength	[5e-4, 4e-4, 3e-4]
Activation Function	[`relu`, `leaky_relu`, `gelu`]
Optimizer	['adam', 'adamw', 'sgd']
Momentum (for SGD)	[0.8, 0.9, 0.99]
Weight Decay (for AdamW)	[1e-5, 1e-4, 1e-3]
CBAM Filters (for ConvNeXtV2)	[32, 64, 128]
Attention	[True, False]
ArcFace Scale (s)	[16.0, 30.0, 48.0]
ArcFace Margin (m)	[0.3, 0.4, 0.5, 0.6]

applying small perturbations to specific numerical hyperparameters. At each iteration, we randomly select one or more hyperparameters to adjust, generating new candidate configurations by adding or subtracting a small step size. During this stage, we train each model for a maximum of 40 epochs, where the candidate configurations are once again evaluated based on Macro-F1 on the validation set. If the new configuration yields an improvement over the current best, it is accepted as the new best configuration. This process continues for 25 iterations. We perform this entire sequence 25 times to ensure a thorough exploration of the local hyperparameter space.

4.3.3.3. Final Model Selection

The best model from each CNN after the SHC stage is selected based on the final Macro-F1 score on the validation set. This model is then used in the final evaluation of all approaches.

4.3.4. Test Selection

With our models selected, we proceed to the test selection phase, wherein each model is executed using our test selection approach outlined in this section. For this, the road data is converted into an image, which is then processed by the CNN to select 'failing' test cases. A diversity-aware selection method is then applied to the pool of selected tests, using visual similarity of test case images as the diversity metric (described in Section 4.3.4.3).

4.3.4.1. PARALLEL IMAGE GENERATION

To improve upon the computational efficiency of the `DRVN` tool, we replaced Matplotlib with cv2 for image generation, and implemented several optimizations. Rather than encoding images as complete image files (*e.g.,* PNG or JPEG), we store them directly as raw NumPy arrays in memory, bypassing the overhead of file I/O operations. Additionally, we leverage parallel processing to generate multiple images concurrently. This approach significantly reduces the time required to process large batches of test cases. Furthermore, this method selectively generates the correct image type for the model. For example, a model trained on the heatmap dataset with a line size of 9 will have this correctly created for prediction.

4.3.4.2. PREDICTION AND SELECTION

Predictions are performed using a batch size of 64, and only test cases whose predicted failure probability exceeds a configurable threshold are considered for selection.

4.3.4.3. ADAPTIVE DIVERSITY SELECTION

For the final selection of test cases, which also takes into account the diversity of each test case, we apply an adaptive diversity-aware selection strategy. This approach balances two competing objectives: prioritizing test cases with high predicted failure probabilities while simultaneously ensuring diversity in the selected test suite to maximize coverage of distinct failure scenarios.

Our diversity calculation during selection is based on visual similarity between test case images. We compute color histograms as efficient feature representations for each test case image. Specifically, each image is down-sampled to 56×56 pixels and converted into an $8 \times 8 \times 8$ histogram across the RGB color space, yielding a 512-dimensional feature vector. We use the Bhattacharyya distance [22] between these histograms, as it operates directly on the CNN's native input space while serving as our selection-time diversity metric, where larger distances indicate more visually diverse test cases.

This approach generalizes across all three image representation types: For the *simple plot* dataset (black line on white background), the histogram captures the spatial structure and line patterns in grayscale intensities, with distinct distributions for roads with different geometric shapes. For the *heatmap* and *points* datasets, the histogram captures the distribution of feature intensities across the RGB channels, where roads with different geometric properties (jerk, chord deviation, path angle) produce distinct color distributions. For example, a road with high jerk values will have elevated red channel intensities in the heatmap, while a road with high chord deviation will have elevated green values. The Bhattacharyya distance effectively captures these distributional differences regardless of the underlying dataset type, making it a dataset-agnostic diversity metric.

The selection process operates in two phases. First, all test cases are ranked by their predicted failure probability. We then apply adaptive diversity filtering to the top 30% of candidates (bounded between 50 and 300 tests to handle varying dataset sizes). This threshold is motivated by the Pareto principle, commonly observed in test case prioritization [229], where a minority of high-ranked tests typically reveal the majority of faults. Focusing diversity computation on this subset balances fault detection effectiveness with computational efficiency.

As key innovation, the diversity threshold adapts based on each test's predicted failure probability, where high-probability tests face a relaxed diversity requirement (threshold scaled by 0.5×), given a better likelihood to reveal defects even if similar to other tests. Conversely, low-probability tests must demonstrate stronger diversity (threshold scaled by 2.0×) for inclusion. This is formalized as:

$$\tau(p) = \tau_{base} \times \left(2.0 - 1.5 \times \frac{p - p_{\min}}{p_{\max} - p_{\min}}\right) \tag{4.2}$$

where $\tau(p)$ is the adaptive threshold for a test with probability p, τ_{base} is the base diversity threshold (default 0.15), and $p_{\min}$ and $p_{\max}$ are the minimum and maximum probabilities in the candidate set. The threshold multiplier ranges from 2.0× for the lowest-probability tests (maximum stringency) to 0.5× for the highest-probability tests (minimum stringency), creating a four-fold variation in diversity requirement.

The constants 2.0 and 1.5 define the adaptive range of the diversity requirement. The multiplier ranges from 2.0× for the lowest-probability tests, to 0.5× for the highest-probability tests, creating a four-fold variation in diversity stringency. This ensures that low-confidence predictions must demonstrate substantial distinctiveness (twice as strict as the base threshold) to justify their inclusion, while high-confidence predictions face relaxed diversity requirements (half the base threshold) since their predicted failure likelihood alone provides strong selection justification. The linear interpolation between these bounds balances the competing objectives of exploiting confident predictions, while maintaining sufficient exploration through diverse lower-confidence tests.

We iterate through candidates in probability-ranked order, ensuring the first (highest-probability) test is always selected. For each subsequent candidate, we compute its minimum Bhattacharyya distance to all previously selected tests. The candidate is added to the selection only if this minimum distance exceeds its adaptive threshold $\tau(p)$. This greedy criterion is formalized as:

$$\text{Select test } t \iff \min_{s \in S} d_B(h(t), h(s)) > \tau(p[t]) \tag{4.3}$$

where S is the set of already-selected tests, d_B is the Bhattacharyya distance, $h(t)$ is the histogram of test t, and $\tau(p[t])$ is the adaptive threshold from Equation 4.2. Our greedy strategy (Equation 4.3) ensures that newly selected tests contribute

meaningful diversity relative to their failure likelihood, preventing the selection of redundant high-probability tests while allowing promising diverse candidates from the broader pool.

In the second phase, remaining lower-probability tests (beyond the top 30%) are evaluated against the already-selected diverse set using the same adaptive filtering mechanism. This two-phase approach efficiently balances computational cost with selection quality, focusing intensive diversity computation on the most promising candidates while still allowing diverse lower-probability tests to enter the final selection if they offer sufficient distinctiveness.

4.4. Empirical Evaluation

We conduct an empirical evaluation to assess the effectiveness of geometry-based CNN models for test case selection (TCS) in self-driving cars. Our evaluation is guided by the following research questions:

RQ1 : *To what extent does dataset choice affect performance of the Convolutional Neural Network (CNN) architectures?*

RQ2 : *Which CNN architectures perform best for test case selection when compared to existing state-of-the-art techniques?*

4.4.1. Dataset

We curated a test dataset consisting of ≈16000 new test cases, generated using SDC-Scissor [25]. By using this tool we are able to generate tests from three test generation tools, AmbieGen [104], Frenetic [39] and FreneticV [40]. We request each tool to generate 1000 test cases per run, for a total of 6 runs per tool, where each tool retains only valid tests.

To ensure consistency with the original dataset used for training, we generated tests using the same parameters as the SensoDat dataset [31], specifically setting the risk factor—how aggressive the agent drives—to 1.5, maximum speed to 120 km/h and the maximum allowed out-of-bounds (OOB) to 0.5m. Each test case is executed in the BeamNG simulation environment [19], using the same lane-keeping SDC model as used in the training data. The outcome of each test case (PASS/FAIL) is recorded based on whether the vehicle successfully maintains lane-keeping throughout the test duration.

This dataset is distinct from the training data used for model development, ensuring an unbiased evaluation for our approach and the baselines.

4.4.2. BASELINE

We compare our CNN-based approaches against several baselines using the sdc-competition platform [28]. This includes the ICST competition winner (`ITS4SDC` [82]) and SBFT competition winner (`RoadGeom` [9]). Additionally we compare against our existing CNN-based approach (`DRVN` tool [17]), for a comprehensive comparison.

4.4.3. IMPLEMENTATION AND PARAMETER SETTINGS

All CNN models are implemented using TensorFlow and Keras trained on a high performance cluster (HPC) using NVIDIA A100 GPUs.

Specific training parameters for each CNN (as discussed in Section 4.3.3.1) are defined in Section 4.5.1, as these are implemented through discovery. Furthermore, the selection threshold, as mentioned in Section 4.3.4.2, was manually evaluated for each model by executing different thresholds for each model in the evaluation tool. During which, we observed the following selection threshold for each model: RES-CNN, GF-CNN, and VGG16 = 0.5, EFNB4 = 0.99, CNV2MSC = 0.4, VGG19 = 0.3.

4.4.4. EVALUATION PROCEDURE

RQ1 : *Dataset Choice Across CNN Architectures* — To answer RQ1, we performed a random search (RS) ablation study, training each of the six CNN architectures on each of the three datasets (simple plot, geometric heatmap, geometric points) with two line thicknesses each (5 and 9 pixels), for a total of six dataset variants per architecture. We conducted 200 random samples from the hyperparameter space for each architecture, training each model for a maximum of 20 epochs and evaluating performance using the Macro-F1 score [94]. We further optimized each architecture/dataset combination using stochastic hill-climbing (SHC) optimization on the best-performing architecture/dataset combination identified from the RS stage. This process was repeated 25 times across 25 iterations of hill-climbing to ensure thorough exploration of the local hyperparameter space for each architecture. We compared the top 10% and bottom 10% of trials across all hill-climbing iterations using the Mann-Whitney U test [137] to determine statistical significance, and computed the Vargha-Delaney $\hat{A}_{12}$ effect size [213] to quantify the magnitude of impact. This systematic ablation allows us to assess how dataset choice influences the performance of each CNN architecture. The results are presented in Section 4.5.1.

RQ2 : *CNN Performance Against Existing Techniques* — To answer RQ2, we selected the best-performing models from RQ1 for each architecture and evaluated them against three baseline methods—`DRVN` [17], `ITS4SDC` [82], and `RoadGeom` [9]—using the sdc-competition platform. Each tool was executed 10 times against our curated test dataset to account for stochasticity in the test selection process. We report the median and interquartile range (IQR) across runs for multiple key metrics:

time-to-fault ratio, fault-to-selection ratio, selection speed, and diversity. To assess statistical significance of performance differences, we applied the Mann-Whitney U test, a non-parametric comparison. For each metric, we conducted pairwise comparisons against the best-performing approach using the Vargha-Delaney $\hat{A}_{12}$ effect size [213], which represents the probability that a randomly selected value from one approach outperforms a randomly selected value from another. We applied Bonferroni correction to control family-wise error rate, adjusting the significance threshold to $\alpha = 0.05/6 = 0.0083$ for the six pairwise comparisons per metric. The results are presented in Section 4.5.2.

We interpret $\hat{A}_{12}$ values as: Negligible ($0.44 \leq \hat{A}_{12} < 0.56$), Small ($0.56 \leq \hat{A}_{12} < 0.64$ or $0.36 < \hat{A}_{12} \leq 0.44$), Medium ($0.64 \leq \hat{A}_{12} < 0.71$ or $0.29 < \hat{A}_{12} \leq 0.36$), or Large ($\hat{A}_{12} \geq 0.71$ or $\hat{A}_{12} \leq 0.29$).

4.5. Results

This section discusses our results for each RQ separately.

4.5.1. RQ1 : Dataset choice across CNN architectures

Table 4.3 summarizes the performance of each architecture across the six datasets—Simple, Heatmap and Geometric plots with either 5 or 9 pixel thickness. The Epochs column indicates the total number of completed training epochs across all 200 runs. The variation (from 1,129 to 2,848 epochs) reflects differences in early stopping behavior and training stability; some runs converged early while others completed all 20 epochs. The table reports median Macro-F1 score with interquartile range (IQR), along with minimum and maximum Macro-F1 scores achieved across all epoch records. By applying statistical analysis on the datasets for each architecture, we can obtain a better understanding of how they perform.

To assess the statistical significance of dataset choice for each architecture, we performed pairwise Mann-Whitney U tests with Vargha-Delaney $\hat{A}_{12}$ effect sizes across all dataset combinations, applying Bonferroni correction ($\alpha = 0.05/15 = 0.0033$) to control for multiple comparisons. The analysis revealed distinct dataset preferences for each architecture.

RES-CNN exhibited a clear preference for heatmap_9, which demonstrated statistically significant superiority over all other datasets with medium effect sizes against simple datasets ($\hat{A}_{12} = 0.68$ vs. simple_5, $\hat{A}_{12} = 0.65$ vs. simple_9, p-value < 0.001), medium effect against points_9 ($\hat{A}_{12} = 0.65$, p-value < 0.001), and small effect against points_5 ($\hat{A}_{12} = 0.62$, p-value < 0.001). Heatmap_9 also outperformed heatmap_5, though with only a negligible effect ($\hat{A}_{12} = 0.54$, p-value < 0.001).

Table 4.3. Random search performance comparison across architectures and datasets (grey cells indicate the dataset with the highest Macro-F1 score for each architecture).

Model	Dataset	Epochs	Med. (IQR)	Min	Max
RES-CNN	simple_5	2,287	0.512 (0.094)	0.281	0.621
	simple_9	2,587	0.519 (0.060)	0.266	0.623
	heatmap_5	2,576	0.539 (0.066)	0.277	0.655
	heatmap_9	2,736	0.548 (0.065)	0.276	0.662
	points_5	2,500	0.530 (0.049)	0.273	0.605
	points_9	2,560	0.523 (0.052)	0.264	0.609
EFNB4	simple_5	2,848	0.617 (0.334)	0.277	0.816
	simple_9	2,662	0.579 (0.331)	0.278	0.809
	heatmap_5	2,714	0.602 (0.319)	0.274	0.829
	heatmap_9	2,640	0.577 (0.312)	0.275	0.828
	points_5	2,813	0.520 (0.326)	0.272	0.803
	points_9	2,570	0.469 (0.306)	0.264	0.804
GF-CNN	simple_5	2,137	0.376 (0.004)	0.280	0.571
	simple_9	2,208	0.379 (0.003)	0.267	0.575
	heatmap_5	2,031	0.380 (0.100)	0.275	0.591
	heatmap_9	2,074	0.379 (0.101)	0.276	0.579
	points_5	2,210	0.383 (0.004)	0.272	0.546
	points_9	2,254	0.387 (0.119)	0.263	0.547
CNV2MSC	simple_5	1,129	0.377 (0.097)	0.276	0.619
	simple_9	2,269	0.385 (0.117)	0.264	0.714
	heatmap_5	2,254	0.282 (0.102)	0.275	0.695
	heatmap_9	2,296	0.340 (0.102)	0.275	0.700
	points_5	1,588	0.288 (0.108)	0.272	0.665
	points_9	2,216	0.270 (0.200)	0.264	0.658
VGG16	simple_5	2,027	0.377 (0.002)	0.373	0.380
	simple_9	2,055	0.387 (0.002)	0.383	0.390
	heatmap_5	1,833	0.374 (0.002)	0.369	0.378
	heatmap_9	2,119	0.380 (0.002)	0.376	0.383
	points_5	1,685	0.374 (0.002)	0.370	0.377
	points_9	1,734	0.374 (0.002)	0.370	0.379
VGG19	simple_5	2,182	0.379 (0.002)	0.375	0.382
	simple_9	2,117	0.378 (0.002)	0.375	0.382
	heatmap_5	2,139	0.380 (0.002)	0.376	0.383
	heatmap_9	2,114	0.380 (0.002)	0.376	0.384
	points_5	2,080	0.382 (0.002)	0.379	0.385
	points_9	2,110	0.388 (0.002)	0.384	0.391

For EFNB4, simple_5 demonstrated modest superiority over several datasets, though predominantly with negligible effects (vs. simple_9: $\hat{A}_{12} = 0.53$; vs. heatmap_9: $\hat{A}_{12} = 0.55$; vs. points_5: $\hat{A}_{12} = 0.55$, all p-value < 0.001), with only points_9 showing a small effect ($\hat{A}_{12} = 0.61$, p-value < 0.001). Similarly, heatmap_5 outperformed points_9 with a small effect ($\hat{A}_{12} = 0.57$, p-value < 0.001) and heatmap_9 with negligible effect ($\hat{A}_{12} = 0.53$, p-value < 0.001). Overall, EFNB4 exhibited relatively weak dataset preferences compared to other architectures.

Conversely, GF-CNN exhibited the strongest dataset preference, with points_5 achieving large effects over simple datasets ($\hat{A}_{12} = 0.76$ vs. simple_5, $\hat{A}_{12} = 0.77$

vs. simple_9, p-value < 0.001) and medium effects over heatmap datasets ($\hat{A}_{12} = 0.70$ vs. heatmap_5, $\hat{A}_{12} = 0.70$ vs. heatmap_9, p-value < 0.001). Points_9 also performed competitively, showing only negligible differences from points_5 ($\hat{A}_{12} = 0.54$, p-value < 0.001).

CNV2MSC displayed high variability and complex dataset interactions. Despite simple_9 achieving the best median score, statistical analysis revealed that geometry-aware datasets generally outperformed simple representations. Specifically, heatmap_5, heatmap_9, and points_5 all demonstrated small effects over simple_9 ($\hat{A}_{12} = 0.56$, $\hat{A}_{12} = 0.57$, $\hat{A}_{12} = 0.58$ respectively, p-value < 0.001), while points_5 showed the strongest overall performance with a medium effect over points_9 ($\hat{A}_{12} = 0.64$, p-value < 0.001). This suggests CNV2MSC benefits from explicit geometric feature encoding despite higher training instability.

The VGG architectures exhibited extreme dataset sensitivity with distinct preferences. VGG16 overwhelmingly favored simple_9, demonstrating large effects against all other datasets ($\hat{A}_{12} = 1.0000$ vs. heatmap_5, heatmap_9, points_5, and points_9, all p-value < 0.001). Notably, heatmap_9 also showed strong performance with a large effect over simple_5 ($\hat{A}_{12} = 0.9808$, p-value < 0.001). Conversely, VGG19 strongly preferred points_9, achieving large effects against all other datasets ($\hat{A}_{12} = 1.0000$ vs. simple_5, simple_9, heatmap_5, and heatmap_9; $\hat{A}_{12} = 0.9997$ vs. points_5, all p-value < 0.001). These findings confirm that dataset choice significantly impacts model performance and that optimal representations vary substantially across architectural families.

> Dataset choice significantly influences CNN performance, with architecture-specific preferences evident across all models. The heatmap and points representations generally outperformed simple plots across most architectures, demonstrating the value of explicit geometric feature encoding.

Following the random search ablation, we performed stochastic hill-climbing optimization to refine hyperparameters for the best-performing architecture/dataset combination of each architecture. Through Mann-Whitney U testing and Vargha-Delaney effect size analysis comparing the top 10% and bottom 10% of trials across 25 iterations of hill-climbing, we identified which hyperparameters most significantly influence each CNN architecture's performance.

Table 4.4 presents the parameter effect sizes (Vargha-Delaney $\hat{A}_{12}$) comparing the top 10% and bottom 10% performing trials during hill-climbing optimization. The best Macro-F1 score achieved during optimization is shown in the bottom row.

Dropout rate and learning rate emerge as the more critical hyperparameters across most architectures. For example, RES-CNN shows a large effect size for dropout rate ($\hat{A}_{12} = 0.18$), indicating that lower dropout values consistently improve performance. Similarly, CNV2MSC exhibits a very large effect size for dropout rate ($\hat{A}_{12} = 0.01$). Learning rate also shows large effects across most architectures, though GF-CNN

Table 4.4. Hill-Climbing parameter effect sizes: Top 10% vs bottom 10% comparison.

Parameter	RES	EFN	GF	CNV2	V16	V19
Learning rate	L	L	L	—	L	—
Dense units	S	M	L	—	M	S
Dropout rate	L	L	L	S	—	L
L2 strength	—	—	L	S	L	L
Weight decay	L	S	—	—	—	—
Momentum	—	—	—	—	M	—
Arcface s	—	S	—	S	—	—
Arcface m	—	S	—	S	—	—
Cbam filters	—	—	—	—	—	—
Best Macro-F1	0.982	0.996	0.719	1.000	0.379	0.379

exhibits only a medium effect ($\hat{A}_{12} = 0.69$), suggesting this architecture is less sensitive to learning rate variations.

Dense units demonstrate varying importance across architectures. GF-CNN shows a large effect size ($\hat{A}_{12} = 0.24$) favoring smaller networks, while EFNB4 exhibits only a small effect ($\hat{A}_{12} = 0.42$), indicating its pretrained weights accommodate various network widths.

Architecture-specific parameters show a much smaller effect overall. For instance, arcface s and m show negligible effects for most architecures that implement it, except for CNV2MSC where arcface m has a small effect size ($\hat{A}_{12} = 0.59$). CBAM filters in CNV2MSC also show negligible effects, suggesting that the attention mechanism's capacity is less critical than other hyperparameters.

Relative to the best random-search result for each architecture, hill-climbing improved Macro-F1 by 79.2% for RES-CNN (0.548 to 0.982), 65.4% for EFNB4 (0.602 to 0.996), 89.2% for GF-CNN (0.380 to 0.719), and 159.7% for CNV2MSC (0.385 to 1.000). The two VGG models did not improve in this refinement stage: VGG16 changed by –2.1% (0.387 to 0.379) and VGG19 by –2.3% (0.388 to 0.379), indicating that hill-climbing did not yield additional benefit for these architectures. Dropout rate and learning rate emerge as the most critical hyperparameters across all CNN architectures, with consistent large effect sizes. Architecture-specific parameters show more limited impact, suggesting that general regularization and optimization settings are more influential than model-specific design choices.

4.5.2. RQ2 : CNN performance against existing test selection techniques

To understand the practical effectiveness of our CNN architectures in selecting test cases for SDCs, we evaluated the top-performing models from our RQ2 optimization

Table 4.5. Test case selection performance reported as median (IQR) across evaluation runs. Diversity is measured as the number of geometric clusters represented by the selected test cases. Gray cells indicate the best result for fault-to-selection ratio and diversity (higher is better), and initialization time, selection time, and time-to-fault (lower is better). Selection reports the number of selected test cases and is not ranked as a performance objective.

Approach	Selection	Init (s)	Select (s)	Time/Fault	Fault/Sel.	Diversity
CNV2MSC	11.50 (10.00)	0.56 (0.04)	26.63 (2.48)	11.71 (3.57)	0.70 (0.13)	5.00 (2.00)
EFNB4	759.50 (48.00)	1.59 (0.12)	28.07 (1.59)	9.59 (0.05)	0.91 (0.02)	8.00 (1.00)
GF-CNN	1277.00 (125.00)	0.20 (0.13)	11.87 (0.29)	12.07 (0.79)	0.77 (0.04)	9.00 (1.00)
ITS4SDC	1007.50 (41.00)	0.00 (0.00)	12.64 (0.39)	9.45 (0.09)	0.93 (0.01)	8.50 (1.00)
DRVN	261.00 (147.00)	0.40 (0.04)	120.40 (40.44)	13.85 (1.33)	0.66 (0.04)	8.00 (2.00)
RES-CNN	906.00 (10.00)	0.18 (0.02)	15.22 (1.27)	12.05 (0.17)	0.76 (0.01)	9.00 (1.00)
RoadGeom	563.50 (5.00)	0.00 (0.00)	3.09 (0.09)	12.92 (1.26)	0.78 (0.07)	8.00 (2.00)
VGG16	5.00 (1.00)	0.26 (0.03)	18.10 (4.47)	9.22 (3.50)	0.82 (0.25)	2.50 (1.00)
VGG19	2.50 (1.00)	0.35 (0.04)	17.91 (3.92)	9.55 (1.19)	1.00 (0.00)	2.00 (0.00)

using the sdc-competition evaluator. As outlined in Section 4.4, we compared our models against our original `DRVN` tool [17], as well as the ICST competition winner (`ITS4SDC`) [82] and SBFT competition winner (`RoadGeom`) [9]. The results are summarized in Table 4.5. We exclude VGG16 and VGG19 from comparative analysis as both models failed to perform adequately during final validation despite showing promise during hyperparameter optimization.

Comparison Against Baselines. When comparing our CNN approaches against the baseline methods, we observe several key findings. For time-to-fault ratio (lower is better), EFNB4 achieved competitive CNN performance (median = 9.59s) but was statistically significantly outperformed by `ITS4SDC` (median = 9.45s, $\hat{A}_{12} = 0.69$, p-value < 0.001, medium effect), indicating that `ITS4SDC` demonstrates superior early fault detection. However, both GF-CNN (median = 12.07s) and RES-CNN (median = 12.05s) performed significantly worse than `ITS4SDC` (large effects, p-value < 0.001). `RoadGeom` also underperformed relative to EFNB4 (median = 12.92s, large effect). For fault-to-selection ratio (higher is better), EFNB4 achieved strong performance (median = 0.91), though `ITS4SDC` significantly outperformed it (median = 0.93, $\hat{A}_{12} = 0.81$, p-value < 0.001, large effect). Both significantly outperformed `RoadGeom` (median = 0.78, large effects, p-value < 0.001). `RoadGeom` dominated in selection speed (median = 3.09s), achieving large effects over all CNNs ($\hat{A}_{12} = 1.00$, p-value < 0.001), with GF-CNN providing the fastest CNN alternative (median = 11.87s). For diversity, GF-CNN and RES-CNN (both median = 9.0 clusters) achieved the highest scores, outperforming `RoadGeom` (median = 8.0, $\hat{A}_{12} = 0.76$, p-value < 0.001, large effect), and outperforming `ITS4SDC` (median = 8.5, $\hat{A}_{12} = 0.61$, p-value $= 0.008$, small effect, Bonferroni-significant at $\alpha = 0.0083$).

Comparison Across CNNs. Among our four CNN architectures, the results reveal distinct performance profiles optimized for different objectives. For time-to-fault ratio, EFNB4 substantially outperformed all other CNNs (median = 9.59s vs. 11.71–12.07s, large effects, p-value < 0.001), demonstrating its superior ability for early fault detection. For fault-to-selection ratio, EFNB4 again led (median = 0.91), significantly outperforming GF-CNN (0.77), RES-CNN (0.76), and CNV2MSC (0.70). In terms of diversity measured by coverage of clusters, GF-CNN and RES-CNN achieved the highest cluster counts (median = 9.0), demonstrating negligible difference between them ($\hat{A}_{12} = 0.50$, p-value = 0.50), with EFNB4 slightly behind (8.0, small effect, $\hat{A}_{12} = 0.62$, p-value = 0.006). CNV2MSC exhibited notably poor diversity (5.0 clusters, large effect vs. GF-CNN: $\hat{A}_{12} = 0.97$, p-value < 0.001), suggesting its selection strategy prioritizes predicted failure probability over coverage of geometrically diverse failure modes. For selection time, GF-CNN achieved the fastest performance (median = 11.87s), followed by RES-CNN (15.22s), CNV2MSC (26.63s), and EFNB4 (28.07s), reflecting the computational overhead of the EfficientNet architecture and the ArcFace margin-based classification head.

When compared against baseline methods, CNN performance is mixed and depends on the evaluation metric. EFNB4 achieves competitive early fault detection (time-to-fault) and strong fault-to-selection ratios, comparable to `ITS4SDC`. GF-CNN and RES-CNN excel in diversity of selected test cases. However, `RoadGeom` demonstrates superior selection speed, and `ITS4SDC` shows marginally better fault-to-selection performance. No single approach dominates across all metrics.

4.6. Threats to Validity

4.6.1. Internal Validity

While we have taken care to explore a wide range of hyperparameters, through random search and stochastic hill-climbing, it is still likely we did not identify the absolute best parameters for each architecture and dataset combination. The VGG16 and VGG19 models' poor performance in final evaluation suggests they may have been particularly affected by suboptimal hyperparameter selection or training instability, given the better performing VGG19 model in the `DRVN` tool.

We performed exploratory analysis on the training dataset to identify geometric features (jerk, chord deviation, path angle) that are predictive of failure. We then encoded these features into the heatmap and points datasets used for model training. This approach is methodologically sound, as exploratory analysis can inform representation design. This approach does not constitute label leakage, as the geometric properties are inherent to the road structure and not derived from the labels. The models must still learn the PASS/ FAIL mapping from these features.

A further ablation study, comparing the road plots, validates that the identified features provide genuine predictive benefit and that no information about the label directly appears in the input data. This approach means we cannot claim to have discovered these features from the CNN models themselves, but instead we validated that a guided representation exploiting known geometric properties supports effective learning. This is a methodological choice, not a flaw, but we make it explicit to avoid overclaiming originality of the feature insights.

Our failure point estimation relies on linear interpolation based on time-to-failure metrics, which implicitly assumes constant vehicle speed throughout the road traversal. This assumption is unrealistic in practice. Autonomous driving systems typically employ deceleration and control instability as they approach failure conditions—sharp curves trigger early braking, instability manifests as oscillations that consume time without covering distance, and loss-of-control scenarios involve erratic acceleration/deceleration patterns. These dynamics mean that time-to-failure does not linearly map to road distance, and our estimated failure points (x_f, y_f) may be systematically mislocated, potentially occurring up or downstream of the actual point of instability. This affects the reliability of our geometric feature analysis: if failure points are inaccurate by several road points, the 'local window of ± 10 points' used to compute feature statistics may systematically sample incorrect neighborhoods, biasing our conclusions about which features (jerk, chord deviation, path angle) are most predictive of failure.

Regarding diversity measurement, each approach employs its own diversity calculation during selection: our CNN models use histogram-based Bhattacharyya distance (efficient, image-space metric), while baselines use their built-in mechanisms. For evaluation, we apply clustering-based measurement on curvature profiles, which is an effective way to understand the semantic uniqueness and separation of selected test cases [64]. We cluster each approach separately due to competition platform constraints—the sdc-competition evaluator assesses tools independently without intermediate diversity access. This per-approach clustering (rather than union-based) ensures fair comparison using consistent methodology (curvature profiles + K-means) while avoiding artificial inflation of diversity scores.

4.6.2. EXTERNAL VALIDITY

Our evaluation is limited to the BeamNG simulation environment with a specific lane-keeping system configuration. Given we are training on road points and their geometric representations, the learned models may still not generalize well to different SDC systems.

To remove the potential of data leakage, we ensured that all approaches were validated and tested on completely new test cases. The training dataset of $\approx 30,000$ test cases and evaluation dataset of $\approx 16,000$ test cases were both generated using SDC-Scissor with three specific test generators (AmbieGen, Frenetic, FreneticV). These generators produce roads with particular geometric characteristics that may

not be representative of real-world road networks or test cases generated by other methods. Our models might perform differently on roads with different complexity distributions or geometric patterns.

4.6.3. Construct Validity

The choice to represent road geometry as a 2D image introduces inherent trade-offs that affect what the models can learn. Converting a 1D ordered sequence of road points into a pixel grid increases input dimensionality substantially—from at most ~ 200 coordinate pairs to a $224 \times 224 \times 3$ tensor—and discards explicit point ordering. As a result, properties such as the direction of travel and the sequential structure of geometric changes are not directly accessible to the CNN. The model must instead infer these from spatial patterns in the rendered image, which may not always recover the same information faithfully. Rendering parameters such as line thickness and the choice of colour encoding can also influence what patterns are visible to the network, introducing a dependency on representation design that does not exist for sequence-based models.

Our primary evaluation metrics (number of faults detected and time-to-fault ratio) assumes that earlier detection of more faults indicates better test case selection (TCS). However, this may not fully capture the practical value of a test suite. Some faults may be more critical than others, and our approach treats all failures equally without considering their severity.

The diversity metric based on visual similarity of road images may not align with semantic diversity of failure modes. Two visually distinct roads might trigger the same underlying fault, while two visually similar roads might reveal different defects. The histogram-based similarity measure captures color distribution differences but may miss subtle geometric patterns that are functionally significant for autonomous vehicle agents.

Regarding surrogate evaluation, we acknowledge that F1-score, may not fully capture surrogate utility for test generation. A surrogate optimizing for precision might miss rare failure modes, while one maximizing recall might generate excessive false positives. We mitigate this by using Macro F1-score, which balances both concerns, and by evaluating a heterogeneous set of architectures to identify consistent patterns independent of specific metrics or model families. All surrogates operate on identical hand-designed features, which could introduce bias toward models that happen to fit those features well. However, this concern extends beyond surrogate design to our core evaluation: the Macro-F1 score used during hyperparameter optimization measures classification performance on historical PASS/FAIL labels but does not directly measure test selection effectiveness. A model with high Macro-F1 during training may not necessarily select the most valuable tests in practice, as evidenced by our observation that GF-CNN and RES-CNN, despite achieving lower Macro-F1 scores than CNV2MSC, demonstrated superior practical performance during test case selection.

4.6.4. CONCLUSION VALIDITY

While we evaluated each approach 10 times to account for stochasticity, our statistical analysis, while thorough, cannot conclusively determine whether observed performance differences between architectures are statistically significant or due to random variation. Furthermore, the random search phase involved 200 iterations per architecture-dataset combination, but successful training varied considerably ($1,129$ to $2,848$ epochs) due to possible early stopping or convergence issues. This imbalance may bias our conclusions about which datasets work best with which architectures, as some combinations were explored more thoroughly than others.

Finally, our evaluation on a single test dataset of $\approx 16,000$ test cases, while substantial, represents a limited sample of possible driving scenarios. Performance rankings between approaches might change with different test distributions or larger evaluation sets.

4.7. CONCLUSION AND FUTURE WORK

In this paper, we presented a comprehensive evaluation of convolutional neural networks (CNNs) for test case selection (TCS) in self-driving cars. A key observation from our study is that the effectiveness of learning-based TCS depends strongly on how road geometry is represented. Our best performing CNN architectures, EFNB4, GF-CNN, and RES-CNN, consistently achieved their strongest results when paired with the heatmap dataset. This supports our hypothesis that the heatmap representation encodes richer geometric information that better captures abrupt changes in road geometry. Although GF-CNN and RES-CNN achieved lower Macro-F1 scores than CNV2MSC during training, they demonstrated superior practical performance during test case selection, highlighting that classification accuracy alone is not a reliable proxy for TCS effectiveness and emphasizing the importance of representation choice when learning from road geometry.

The strong performance of the heatmap representation, particularly in combination with EFNB4, also supports the motivation for evaluating architectures sourced from the SnakeCLEF competition. The geometric heatmap produces banding-colored, continuously-varying curves on a plain background—properties that are visually similar to the snake images used in the SnakeCLEF competition [167]. This visual resemblance, together with the strong inductive biases of EfficientNet-B4 and ConvNeXtV2 (compound scaling, depthwise convolutions, multi-scale fusion), provides a principled basis for their inclusion in the evaluation.

Despite not achieving the best performance in our evaluation, many of the CNN architectures we evaluated, particularly the geometry-aware models, demonstrated competitive performance against strong baselines sush as `ITS4SDC`. This suggests that CNNs have potential as effective surrogate models for test case selection for SDCs, especially when paired with appropriate geometric representations.

Therefore, future work could explore additional CNN architectures, such as vision transformers or graph neural networks, which may better capture the complex spatial relationships in road geometry. Additionally, further investigation into representation design, including alternative geometric encodings, or mulit-modal representations that combine geometeric features could yield further improvements in test case selection performance.

5

THE LAST DEPENDENCY CRUSADE: SOLVING PYTHON DEPENDENCY CONFLICTS WITH LLMS

Resolving Python dependency issues remains a tedious and error-prone process, forcing developers to manually trial compatible module versions and interpreter configurations. Existing automated solutions, such as knowledge-graph-based and database-driven methods, face limitations due to the variety of dependency error types, large sets of possible module versions, and conflicts among transitive dependencies. This paper investigates the use of Large Language Models (LLMs) to automatically repair dependency issues in Python programs. We propose `PLLM` (pronounced "plum"), a novel retrieval-augmented generation (RAG) approach that iteratively infers missing or incorrect dependencies. `PLLM` builds a test environment where the LLM proposes module combinations, observes execution feedback, and refines its predictions using natural language processing (NLP) to parse error messages. We evaluate `PLLM` on the Gistable HG2.9K dataset, a curated collection of real-world Python programs. Using this benchmark, we explore multiple PLLM configurations, including six open-source LLMs evaluated both with and without RAG. Our findings show that RAG consistently improves fix rates, with the best performance achieved by `Gemma-2 9B` when combined with RAG. Compared to two state-of-the-art baselines, `PyEGo` and `ReadPyE`, `PLLM` achieves significantly higher fix rates; +15.97% more than `ReadPyE` and +21.58% more than `PyEGo`. Further analysis shows that `PLLM` is especially effective for projects with numerous dependencies and those using specialized numerical or machine-learning libraries.

This chapter was presented at the first annual Agentic Software Engineering Workshop (AgenticSE), co-located with ASE'25 in Seoul, Republic of Korea by Antony Bartlett, Cynthia C. S. Liem and Annibale Panichella [14]. See Chapter 1.6 for additional publication details.

5.1. Introduction

Python, introduced in 1991, has become one of the most widely used programming languages [38], due in part to its extensive ecosystem of reusable modules. These modules, once imported, become project dependencies that must be managed over time. As Python's ecosystem grew, so did the complexity of dependency management, prompting the development of tools such as `distutils` [186], `setuptools` [2], and `pip` [1], which are based on the Python Package Index (`PyPI`) to install direct and transitive dependencies.

Yet despite these advances, many Python programs still fail to run out-of-the-box [98,226], typically due to missing, incompatible, or conflicting dependencies. A *dependency conflict* occurs when two or more modules require different and incompatible versions of the same dependency. This can arise directly—when a specified version is unavailable or mismatches another requirement—or transitively, when nested dependencies bring incompatible constraints. Such conflicts often result in runtime errors, broken APIs, or failed installations. This problem is particularly acute in machine learning and scientific computing, where libraries frequently depend on hardware-specific versions (*e.g.,* CUDA) or require tightly coupled version combinations [102]. Even minor version updates can introduce breaking changes [63], highlighting the need for precise and automated dependency resolution. Existing automated solutions rely on knowledge graphs (*e.g.,* `PyEGo` [227], `ReadPyE` [45]), but these struggle with complex dependencies and require frequent updates.

We introduce `PLLM`, an LLM-based method that resolves dependency conflicts through iterative repair. `PLLM` combines Retrieval-Augmented Generation with build feedback, addressing hallucinations through concrete error feedback [177]. To explore this, we introduce `PLLM` (pronounced "plum"), a novel LLM-based method that automatically resolves dependency issues through an iterative repair process.

We evaluate `PLLM` on the Gistable HG2.9K, a curated benchmark of real-world challenging Python programs commonly used to assess dependency resolution methods. This benchmark is widely used in the literature to evaluate the performance of dependency conflict resolution techniques [45,99,227]. The following research questions guide our study:

RQ1 : *To what extent can current LLMs infer working dependencies from a given Python file?*

RQ2 : *How does `PLLM` compare to state-of-the-art knowledge-based dependency resolution techniques?*

RQ3 : *Under what conditions does PLLM outperform traditional methods?*

We evaluated PLLM with six LLMs, observing the use of RAG as the most performant. Once evaluated against the entire HG2.9K, PLLM was found to significantly outperform knowledge-graph baselines PyEGo and ReadPyE, achieving +21.58% and +15.97% more fixes respectively.

PLLM is particularly effective for projects with intricate dependency structures, excelling at complex dependencies such as those for machine learning (*e.g.,* tensorflow) and numerical computing (*e.g.,* scipy), contributing the most unique conflict resolutions overall.

5.2. Background and Related Work

5.2.1. Existing Approaches

Knowledge graphs have been widely used to encode relationships among Python modules. Horton and Parnin [99] introduced DockerizeMe, which constructs a knowledge graph using Libraries.io data stored in Neo4J to infer required dependencies and resolve transitive dependencies.

Building on this work, PyEGo [227] expands DockerizeMe with package documentation and release metadata, using 256,000 nodes and 1.9 million relationships.

ReadPyE [45] combines naming similarity with optimization algorithms for module matching. By iteratively validating dependency choices and adjusting module selections ReadPyE finds solutions based on validation logs. Although this method improves matching accuracy, it and PyEGo require regular updates to maintain the graph's relevance, a common limitation among knowledge-graph-based tools.

In contrast to graph-based methods, Mukherjee *et al.* [150] proposed PyDFix, which uses regex-based parsing of Python error logs to infer missing dependencies. By analyzing build failures and iteratively patching the environment, PyDFix provides a lightweight, runtime-driven alternative. However, regex-based methods are inherently brittle: even small changes in log formatting can break the parsing logic, limiting the generality and robustness of such approaches.

While these methods have contributed significantly to the field, they suffer from key limitations: knowledge graphs require large databases and frequent updates, while regex methods are brittle to format changes [139,233,237].

PLLM uses an LLM within a RAG pipeline to dynamically resolve dependencies by retrieving PyPI metadata and incorporating build error feedback, avoiding static infrastructure and reducing model hallucination risks [177]. This adaptive pipeline reduces the need for pre-built infrastructure and improves generalization across a broad range of dependency error types.

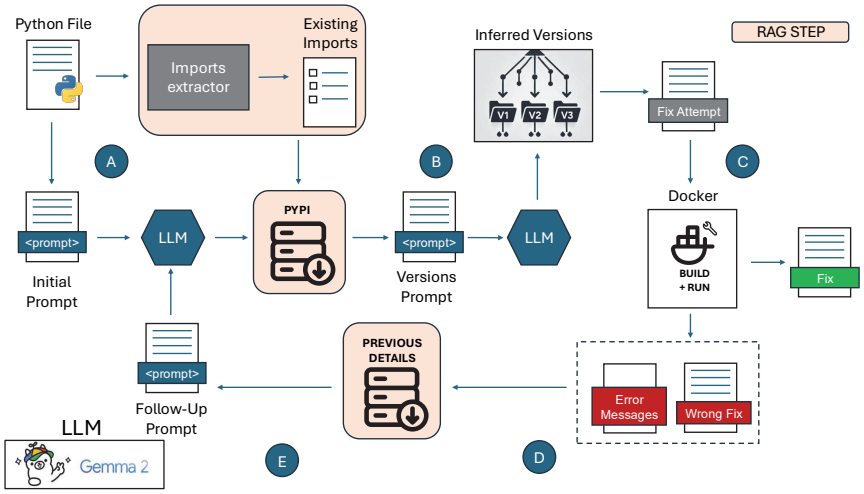


Figure 5.1. Overview of PLLM, which encompasses five main stages: (A) extracting import statements from the input Python file, (B) prompting the model to infer modules and Python versions, (C) generating candidate dependency fixes using PyPI metadata, (D) validating candidate fixes through Docker build and execution, and (E) providing error-based feedback to the LLM for iterative refinement.

5

5.2.2. Datasets of Python Dependency Conflicts

Evaluating dependency repair techniques requires datasets that include real-world Python code with known dependency issues. The most widely used benchmark is the Gistable dataset, introduced by Horton and Parnin [98]. This dataset, collected from GitHub's Gist platform[1], contains over 10,000 Python gists—programs that vary in complexity. The authors found that only 24.4% of these gists could be executed without modification, while 52% failed due to missing imports. The Gistable dataset[2] includes a challenging subset of 2,891 "hard" gists that fail due to missing imports and remain unrunnable without fixes. This subset has become the gold standard for evaluating dependency-fixing approaches.

5.3. Our Approach

PLLM is a prompt-based LLM system supporting multiple open-source backends. Figure 5.1 illustrates our five-stage pipeline combining RAG and LLMs for iterative Python dependency resolution.

- *Stage A*: Inferring module names and Python version.

[1] https://gist.github.com/
[2] https://github.com/gistable/gistable

- *Stage B*: Inferring module versions.
- *Stage C*: Docker-based build and validation.
- *Stage D*: Error analysis and classification.
- *Stage E*: Feedback loop and iterative refinement.

We now detail each stage, describe the prompts used, and explain how RAG contributes to each step.

5.3.1. Inferring Module Names and Python Version (Stage A)

`PLLM` prompts the LLM with the input Python file to infer required modules and Python version (Listing 5.1). Placeholders like `raw_file` and `format_instructions` (JSON schema in Listing 5.2) make prompts modular and reusable.

When RAG is enabled, `PLLM` complements the LLM output by running a regex-based static analysis to extract all import statements from the input. These are merged with the LLM-inferred modules to form a more complete list of module/dependencies to consider for conflict fixes. We further elaborate on RAG in Section 5.3.5.1.


```
Prompt:
"Given a python file:\n{raw_file}\nReturn a list of Python modules and python version
    ↪ required to run. Output JSON based on the schema {format_instructions}"
```

Listing 5.1 Prompt for inferring information from a given Python file.

5.3.2. Inferring Module Versions (Stage B)

Stage B prompts the LLM to assign versions to each identified module. Using RAG, we provide PyPI metadata to avoid hallucinated versions (Listing 5.3); without RAG, the LLM infers versions from context (Listing 5.4). Standard library modules are filtered out, and import/install name mismatches are resolved via curated mapping, *e.g.,* `sklearn` → `scikit-learn`, `bs4` → `beautifulsoup4` as shown in Listing 5.5.

The prompt with RAG in Listing 5.3 contains a larger set of placeholders, as we provide the LLM with specific dependency information for guidance. In the Listing, `'module_name'` represents the name of the module (dependency) obtained from the initial prompt (Stage A). When RAG is enabled, we provide `'module_versions'`, a comma-separated list of available versions sourced from PyPI, for the given dependency. We also provide the LLM with a list of previously attempted versions `'previous_versions'` to prevent using versions that already resulted in unsuccessful conflict fixes. To promote diversity and avoid local optima, we request the LLM to sample versions evenly across the available list. Finally, the

```
{ "python_modules": [{"module": "<String>", "version": "<String>"}], "python_version":
    "<String>" }
```

Listing 5.2 JSON Schema: Dependencies and Python version.

```
Prompt:
"Given a comma-separated list of 'Module versions' for the '{module_name}' module,
    ↪ from oldest to newest:\n{module_versions}\nPerform equally distanced sampling
    ↪ to return a version from the given versions, excluding previously used
    ↪ versions ({previous_versions}). Return the information with the format {
    ↪ format_instructions}"
```

Listing 5.3 Prompt with RAG for selecting a version from available PyPI releases.

```
Prompt:
"Infer a possible working version of the '{module_name}' module for Python {
    ↪ python_version}.\nReturn the information with the format {format_instructions
    ↪ }"
```

Listing 5.4 Prompt without RAG (LLM infers version from context).

```
import sklearn
Must be installed as 'scikit-learn'
```

Listing 5.5 Discrepancy between import and install names.

'format_instruction' prompts the LLM to return both the dependency name and the newly chosen version. When RAG is disabled, Stage B uses a much simpler prompt. An example of such a prompt can be seen in Listing 5.4. In this prompt, we only give the LLM the module name, python version and format instructions, allowing it to infer a version only with this information.

5.3.3. Docker-based build and validation (Stage C)

PLLM validates predictions by building and executing the Python file in Docker containers. We construct Dockerfiles with the inferred Python version and modules (Listing 5.6), using unique names for concurrent execution. Success is indicated by a valid configuration; failure triggers error analysis.

5.3.4. Error analysis and classification (Stage D)

Stage D analyzes build/execution errors to guide LLM corrections. Building on prior work in automated environment inference [45,99,227], this hybridized

```
FROM python:3.6
WORKDIR /app
RUN ["pip","install","--upgrade","pip"]
RUN ["pip","install","--trusted-host","pypi.python.org","--default-timeout=100","
    ↪ keras==2.0.9"]
RUN ["pip","install","--trusted-host","pypi.python.org","--default-timeout=100","
    ↪ tensorflow==2.4.4"]
COPY snippet.py /app
CMD ["python", "/app/snippet.py"]
```

Listing 5.6 Dockerfile.

approach reduces over-prompting–the bottleneck of PLLM–by identifying eight common error types that arise during dependency resolution: `VersionNotFound`, `DependencyConflict`, `ImportError`, `ModuleNotFound`, `AttributeError`, `InvalidVersion`, `NonZeroCode`, and `SyntaxError`.

A key insight from our development is that combining multiple failure signals into a single prompt often leads to LLM confusion or hallucinations. Therefore, we adopt a *multi-step prompting strategy* that isolates specific messages and crafts focused prompts, improving interpretability and response accuracy.

For example, `ImportError`, the most common error, is handled with a specialized prompt to identify the missing module (Listing 5.7).

An `ImportError` indicates a missing module that was not installed. We provide the LLM with the error, requesting it return the offending module name. This ensures the LLM handles noisy logs and correctly extracts the relevant information. Stage D thus acts as the diagnostic core, linking runtime feedback to model-driven correction.

5.3.5. Feedback Loop and Iterative Refinement (Stage E)

Stage E uses error feedback to iteratively refine dependency suggestions. Each cycle attempts to repair by inferring (i) missing modules, (ii) different versions, or (iii) additional dependencies until success or a predefined iteration limit is reached. We maintain a history of previously attempted combinations to prevent redundant retries while continuing to verify the existence and naming of modules on PyPI. This stage is imperative, as it also allows PLLM to indirectly locate and address transitive dependencies.

5.3.5.1. Retrieval-Augmented Generation (RAG) Implementation

Our RAG implementation addresses LLM hallucination by retrieving real-time PyPI metadata (550,000+ modules). However, raw metadata can contain considerable unwanted information, that can mislead models. To address this, we filter the metadata to within the release window of the Python version being used, as well as

```
Prompt:
"Given the following ImportError:\n{error_msg}\nIdentify the module causing the error
    ↪ .\nThe module is usually mentioned in a statement like 'from x import y'.\
    ↪ nReturn just the module name using the format {format_instructions}"
```

Listing 5.7 Prompt for identifying the missing module from an ImportError.

the explicit Python version support declared by the module. This ensures that only relevant versions are considered, reducing noise and improving inference accuracy. Filtered metadata is provided as comma-separated version lists, which proved most effective for LLM prompting. Furthermore, in situations where no suitable versions are found, we fall back to the latest release of the module.

5.3.6. Parallel Multi-Version Execution

A distinctive feature of PLLM is its capacity to validate against multiple Python versions simultaneously. For example, if the LLM predicts Python 3.5 and we have allowed for a range of 2, we execute against versions 2.7, 3.4, 3.5, 3.6, and 3.7 in parallel, increasing success likelihood across deployment scenarios.

5.4. Experimental Setup

The *goal* of this study is to evaluate the effectiveness of PLLM in resolving Python dependency conflicts for real-world Python programs. To this aim, we formulated the following research questions:

RQ1 : *To what extent can current LLMs infer working dependencies from a given Python file?*

RQ2 : *How does PLLM compare to state-of-the-art knowledge-based dependency resolution techniques?*

RQ3 : *Under what conditions does PLLM outperform traditional methods?*

RQ1 evaluates current LLM capabilities for dependency inference. **RQ2** and **RQ3** compare PLLM against knowledge-based approaches to determine contexts where PLLM excels while identifying potential limitations.

5.4.1. LLM Model Selection

To ensure openness and reproducibility in our approach, we evaluated six open-source LLMs via Ollama[3]: three `Gemma2` [70] versions (3B, 9B, 27B), `DeepSeek-R1` [83], `Llama3.1` [79], and `MistralAI` [110].

[3] https://ollama.com

5.4.2. Dataset

We validated our approach using the HG2.9K dataset, which has become the standard benchmark for evaluating Python dependency resolution tools [45,227]. This dataset, originally created as part of the `DockerizeMe` paper by Horton and Parnin [99], is a curated subset of the broader Gistable corpus [98]. It contains 2,891 "hard" Gists—Python programs collected from GitHub's Gist platform—that still fail with `ImportError` even after applying Gistable's naïve resolution strategy. As such, it represents realistic and challenging dependency drift scenarios in Python.

To answer **RQ1**, we created a subset of the HG2.9K to reduce computational cost while preserving result validity. The subset, containing 422 randomly selected Gists, was executed five times per LLM with different seeds to account for stochasticity [177]. By using power analysis [207], we yield a confidence level of 95% and a margin of error below 4.5% for binary outcomes. For **RQ2** and **RQ3**, we evaluated the best `PLLM` configuration (as identified in **RQ1**) and the baselines on the full HG2.9K dataset to ensure a comprehensive comparison.

5.4.3. Implementation and Parameter Settings

We implemented `PLLM` using Python 3.11, Langchain 0.2.1, and Ollama 0.2.0. Experiments ran in docker containers on an an AMD EPYC 7713 64-core Processor running at 2.6 GHz (256 CPUs) with an available Nvidia A40 GPU (48GB GDDR6 memory).

Our implementation supports a set of configurable parameters to optimize evaluation and test execution. The key parameters relevant to this study include: `--model=gemma2`, `--temp=0.7`, `--loop=10`, `--range=1`, and `--rag=True`.

The `model` parameter specifies which LLM backend to use. This can be set to any model available in the Ollama model library[4]. For this study, we primarily used `Gemma2`, though other models were evaluated as discussed in Section 5.4.1.

The `temp` parameter controls the temperature for the LLM, with higher values (up to 1.0) encouraging more creative responses. We set this to 0.7 to strike a balance between determinism and variation in output. While the default temperature is not standardized—OpenAI's GPT-4 technical report [158] uses 0.6 arbitrarily, and Ollama's documentation[5] lists both 0.7 and 0.8 as defaults—we selected 0.7 based on its initial status as Ollama's default during early development of `PLLM`.

As described in Section 5.3.3, `PLLM` uses an iterative loop to refine dependency suggestions. We set the `loop` value to 10 based on empirical trade-offs between runtime and success rate. Lower values led to incomplete resolutions, while higher values increased runtime with diminishing returns.

[4]https://ollama.com/library
[5]https://github.com/ollama/ollama/blob/main/docs/modelfile.md

The `range` parameter defines how many Python versions are validated in parallel (see Section 5.3.6). A value of 1 triggers validation across three versions (*e.g.,* v−1, v, v+1), offering both breadth and efficiency.

Lastly, the `rag` flag controls whether Retrieval-Augmented Generation is enabled. It is set to `True` by default but was explicitly disabled when evaluating **RQ1** to isolate the performance of the LLMs without external metadata assistance.

5.4.3.1. Baseline settings

We evaluated `PLLM` against two state-of-the-art baselines introduced in prior work: `PyEGo` [227] and `ReadPyE` [45] (see Section 5.2). Both baselines were executed using the default settings recommended in their respective repositories. Each required Docker-based execution and the creation of a Neo4j knowledge graph. `PyEGo` provided a database dump in its repository, while `ReadPyE` offered this as a separate artifact. Two knowledge graphs are available in `ReadPyE`; for consistency, we selected **KG0**, which focuses solely on Python module inference. This aligns with the scope of `PLLM`, which does not attempt to resolve OS-level dependencies. It is worth noting that `PyEGo` does fix both Python and system-level dependencies, but for evaluation purposes, we restrict comparisons to Python modules only.

5.4.4. Evaluation criteria

To answer **RQ1**, we performed a comparative analysis of various LLMs to understand their performance within the `PLLM` workflow. As part of this analysis, we executed these models with RAG enabled and disabled to understand the effects of RAG in the pipeline. We executed each model a total five times against a subset of the HG2.9K dataset, calculating the average number of fixes discovered.

The top-performing model from **RQ1** was then used in **RQ2** to compare `PLLM` against the two baselines, `PyEGo` and `ReadPyE`, across the full HG2.9K dataset. Following the evaluation criteria established in DockerizeMe [99], we define a successful fix as a Python program that executes without critical runtime errors such as `ImportError`, `Module-NotFoundError`, `AttributeError`, or `SyntaxError`.

For each approach, we also recorded the time taken to infer a working environment, as we were particularly interested in understanding the efficiency of each method in resolving dependency conflicts. Accurate timing information provides insight into the practical usability of these approaches, particularly in scenarios where dependency resolution speed is crucial. Some adjustments were required to obtain timings for our baselines. For `ReadPyE`, more extensive modifications were necessary to achieve precise timing measurements. First, we recorded the inference time, which includes the stage where a Dockerfile was created with the inferred Python modules. Next, we validated the Dockerfiles, as no logging was initially provided to confirm whether a Dockerfile was executable. Each Dockerfile was built and run at this stage, analyzing the output and logging the time required for completion. We then

combined these timings to determine which Dockerfiles were successfully executed. For `PyEGo`, we could infer considerable information from their output logs, which contained various data points, including the start and stop times for each program inference and whether the inference was successful. To account for the stochastic nature of LLMs, we executed `PLLM` 10 times, aggregating the data to understand the full extent of fixes. However, as both of our baselines utilize a static knowledge graph, these approaches only required a single run.

To answer **RQ\3**, we conducted a permutation test [52], a non-parametric alternative to the ANOVA test [190], that does not require the data to be normally distributed. This test allowed us to determine whether there is a significant difference in the number of successful fixes produced by `PLLM`, `PyEGo`, and `ReadPyE`. We also assessed the impact of various project co-factors on these outcomes. The permutation test provided insights into the statistical significance of differences in successful fixes among the approaches without relying on the normality assumptions of ANOVA. To further investigate specific patterns, we analyzed the frequency of successful fixes for particular types of modules (*e.g.,* PyTorch) and for projects with varying numbers of dependencies. We examined the success frequencies across Python files with similar dependencies and different dependency counts. Additionally, we analyzed the intersection and unique sets of projects successfully fixed by `PLLM`, `PyEGo`, and `ReadPyE`, identifying both shared and unique successes for each approach.

The complete replication package, including the datasets, the source code, and results, is available on Figshare[6].

5.5. Empirical Results

In this section, we discuss the results of the experiment with respect to our research questions.

5.5.1. Results for RQ1

Table 5.1. Total number of successful fixes for each of the models with RAG on and off.

RAG	G2-S	G2	G2-L	DS	L3.1	M
Enabled	86	203	187	166	197	194
Disabled	-	171	207	164	111	106

Note: G2-S: Gemma2-2B, G2: Gemma2-9B, G2-L: Gemma2-27B, DS: DeepSeek-R1-8B, L3.1: Llama3.1-8B, M: MistralAI

Table 5.1 indicates comparable performance across most models (∼200 fixes), except `Gemma2-2B` which achieved less than half. RAG improved performance for most

[6] 10.6084/m9.figshare.29204693

Table 5.2. Distribution of successful fix counts per Gist in the HG2.9K subset dataset.

Model	RAG	5	4	3	2	1
`DeepSeek-R1-8B`	✓	80	55	36	36	30
`Gemma2-2B`	✓	33	31	21	22	38
`Gemma2-9B`	✓	166	22	12	19	25
`Gemma2-27B`	✓	146	28	17	10	23
`Llama3.1-8B`	✓	103	53	38	39	35
`MistralAI`	✓	122	49	23	31	35
`DeepSeek-R1-8B`	✗	75	53	43	36	33
`Gemma2-2B`	✗	-	-	-	-	-
`Gemma2-9B`	✗	119	33	18	23	27
`Gemma2-27B`	✗	165	21	19	25	21
`Llama3.1-8B`	✗	49	25	35	23	61
`MistralAI`	✗	53	24	25	31	33

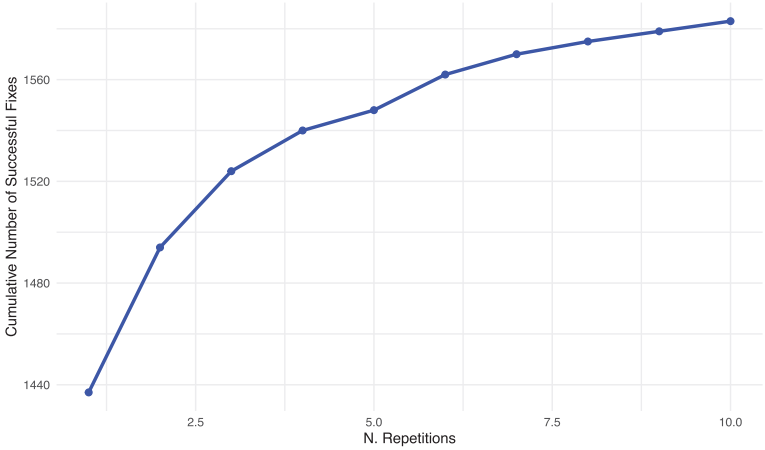


Figure 5.2. Cumulative chart displaying the number of fixes found by `PLLM` across multiple runs.

models, though `Gemma2-27B` performed slightly better without RAG. Considering both performance and consistency (Table 5.2), we selected `Gemma2-9B` with RAG for subsequent experiments. This model provided an optimal balance between high success rates and computational efficiency, showing consistent performance across multiple runs, feasible for extensive evaluation.

Figure 5.2 shows the cumulative fixes for the `Gemma2-9B` model against the entire HG2.9K, over ten independent runs. A single run achieves ∼1,440 fixes, with ad-

ditional runs uncovering more unique solutions due to LLM stochasticity. Even with a single run, PLLM outperforms both baselines (see more details in the next subsection).

Table 5.3. Number successful and unsuccessful fixes produced by PLLM, PyEGo and ReadPyE. We also report the average time (in seconds) needed to generate the successful fixes alongside the interquartile range (IQR).

Method	# Fixed	# Unfixed	Fix Time (IQR)
PLLM	1583	1308	151.461 (221.30)
ReadPyE	1365	1526	62.77 (40.57)
PyEGo	1302	1589	4.025 (7.56)

5.5.2. Results for RQ2

Table 5.3 reports the number of successful and unsuccessful fixes produced by PLLM, PyEGo, and ReadPyE, and the average time (in seconds) required to generate successful fixes, along with the interquartile range (IQR). As observed, PLLM outperforms both baselines in terms of the number of successful fixes, achieving a total of 1583 successful fixes. This is significantly higher than the 1365 and 1302 successful fixes produced by ReadPyE and PyEGo, respectively.

Compared to the baselines, average fix time is higher for PLLM, with an average time of 151.46 seconds (compared to 62.77 seconds for ReadPyE and 4.025 seconds for PyEGo). The higher IQR for PLLM indicates a broader range of fix times, largely due to the iterative nature of our approach, which continues searching until a successful set of dependencies is found. We argue that this additional time is justified by the higher success rate achieved by PLLM, and an average time of two minutes remains reasonable for achieving a successful fix with Dockerfile generation and execution.

To further analyze the effectiveness of our approach, we examined cases where PLLM succeeded while both baselines failed. Figure 5.3 visualizes the overlap of successful fixes across all three approaches, illustrating the unique effectiveness of PLLM. Specifically, all three approaches were successful on 53% of all Gists, while PLLM uniquely succeeded on 13% (representing 237 Gists) that neither baseline could fix. There is a distinct overlap of 981 Gists, representing cases where all three approaches were successful. The combined success across all approaches reaches 1858 Gists. Considering these 1858 as the total potentially fixable Gists within our validation system, PLLM successfully fixed 85% of the fixable Gists in the dataset. This finding strongly supports the effectiveness of our approach in reliably inferring working dependencies from a Python file and highlights the potential of hybrid approaches, which are part of our future agenda.

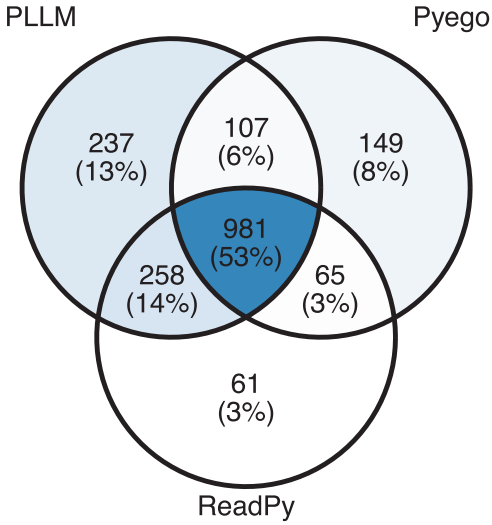


Figure 5.3. Intersections and difference sets for the projects successfully fixed by `PLLM`, `PyEGo`, and `ReadPyE`.

Table 5.4. Percentage of successful fixes produced by `PLLM` and `PyEGo` for projects with a different number of dependencies. The best values are highlighted in gray color

# Project Dependencies	`PLLM`	`PyEGo`	`ReadPyE`
0-2	0.5478	0.4846	0.5046
3-4	0.5500	0.4162	0.4517
5-6	0.5410	0.2551	0.2806
7+	0.5333	0.3555	0.2222

5.5.3. RESULTS FOR RQ3

Permutation tests confirm `PLLM`'s statistical superiority ($p < 0.01$) and significant interaction with dependency count. Table 5.4 shows `PLLM` maintains consistent success (53-55%) regardless of dependency count, while baselines degrade from 50% to 32% (`PyEGo`) and <25% (`ReadPyE`), indicating knowledge-graph methods struggle with complex projects.

Table 5.6 displays examples of Gists in which the various approaches passed and failed. We focus here on examples where `PLLM` was either better or worse than both baselines (*e.g.,* `PLLM` passed, `PyEGo`, `ReadPyE` failed, *etc.*). This allows us to dig deeper into the results of these specific data points to understand why the LLM

Table 5.5. Percentage of successful fixes for the top-15 most frequent Python modules in our dataset for `PLLM`, `PyEGo`, and `ReadPyE`.

Module Name	**# Projects**	`PLLM`	`PyEGo`	`ReadPyE`
numpy	572	69.06	57.17	53.50
django	382	86.65	65.18	79.84
scipy	296	79.39	58.78	62.50
requests	168	66.67	50.60	52.38
pillow	166	76.51	69.88	71.08
matplotlib	162	64.81	62.96	54.94
tensorflow	140	83.57	51.43	62.14
scikit-learn	136	76.47	55.88	66.91
tensorflow-gpu	135	82.96	52.59	68.15
opencv-python	85	64.71	49.41	24.71
pandas	73	68.49	63.01	57.53
cython	69	55.07	33.33	39.13
image	69	23.19	18.84	18.84
pyyaml	69	66.67	66.67	71.01
theano	67	68.66	26.87	37.31
keras	63	77.78	14.29	42.86

Table 5.6. Example Gists showing where `PLLM`, `PyEGo`, and `ReadPyE` succeeded or failed in dependency resolution. Each cell lists inferred dependencies.

Gist ID	`PLLM`	`PyEGo`	`ReadPyE`
019fd5c706e0bc94879f	rx;twisted	rx;twisted	twisted;urx
477a9dcd198439ef2def	urllib2;readability	readability-lxml	readability-lxml
3153844	graphite	NA	graphiti
4e5035242b8e4b07ff3a	pymongo;pip;bson	pymongo	pymongo
c2dfe5772ba3cd16c1be17ba42b7db66	keras;tensorflow	keras	keras;tensorflow-gpu

Note: Grey cells denote successful runs. These cases illustrate complementary strengths and limitations of the different approaches.

succeeds or fails.

In Gist `'019fd5c706e0bc94879f'`, `PLLM` and `PyEGo` both identified `rx` and `twisted` as required modules. In contrast, `ReadPyE` incorrectly inferred `urx`, likely due to a semantic match in its knowledge graph. This resulted in an `ImportError` and failure. While `PyEGo` selected the correct modules, its choice of `rx` version also failed at runtime. All three approaches selected the same version of `twisted`. `PLLM` required a few iterations to converge on a compatible version of `rx`, ultimately producing a successful configuration.

Gist `'477a9dcd198439ef2def'` is an example of where `PLLM` was unsuccessful. By observing the dependency chosen by both baselines, we identify that `readability`

is likely no longer available or has been replaced with `readability-lxml`. We can also observe the same scenario with `'7030355'` and the `mpd`, now `python-mpd2` dependency. These examples could be added to our JSON file mentioned in Section 5.3.2. We also note a failure for `PLLM` with Gist `'4e5035242b8e4b07ff3a'`. This is a little more interesting as the LLM has `pymongo` listed as a dependency. However, unlike the other approaches, it has also attempted to install `pip` and `bson`, which have likely contributed to the failure in this instance.

Finally, we examine Gists `'3153844'` and `'c2dfe57-72ba3cd16c1be17ba42b7db66'` in which `PLLM` was the only successful approach. Here, we witness another example of `ReadPyE` attempting a dependency with an alternative name. In this instance, the original dependency `graphite` was successfully installed by `PLLM`, and no dependency was even attempted by `PyEGo`. The final Gist `'c2dfe5772ba3cd16c1be17ba42b7db66'` has overlap between the baselines, but ultimately, the lack of `tensorflow` in `PyEGo` and the incorrect `tensorflow` dependency for `ReadPyE` led to unsuccessful conflict resolutions for both approaches.

5

These examples illustrate several core strengths of `PLLM`: iterative refinement, flexible version search, and responsiveness to real-time failure signals. However, they also expose failure modes tied to naming mismatches and unnecessary package inference—areas where targeted prompt tuning or expanded mapping rules could yield further gains.

5.6. Threats to Validity

5.6.1. Construct Validity

A key aspect of our approach (`PLLM`) is allowing the LLM to dynamically infer Python versions from Python code and validate these along with adjacent Python versions. This flexible strategy addresses limitations in prior work [45], which rely on static Python versions during testing, potentially limiting both the adaptability and real-world relevance of the results. The only exception in our approach is Python 2.7, which is consistently validated due to its unique, legacy-specific versioning requirements.

Our method ensures that, once execution begins, the Python version remains unchanged, even if a `SyntaxError` occurs—an indicator of an incompatible Python version. Unlike approaches that dynamically switch versions mid-execution, our approach simulates real-world constraints more accurately.

5.6.2. Internal Validity

For evaluation, we used a single LLM, Gemma2 [70], which showed strong performance in tests compared to other open-source LLMs. A small manual study was

performed using the commercial Claude 4 Sonnet model[7]. Claude fixed 15 PLLM-failure cases, already handled by the baselines, but failed on all PLLM-only successes particularly with 3+ dependencies, often hallucinating versions or misusing import names. This supports our open-source choice.

To address the stochastic nature of LLM outputs, we followed existing guidelines [177] on assessing LLMs and executed `PLLM` multiple times, aggregating the results to ensure a more comprehensive assessment of its effectiveness. The model size also impacts internal validity; we selected the 9-billion-parameter version of Gemma2, compatible with both our local machine and the validation servers.

5.6.3. External validity

Our evaluation is based on the HG2.9K dataset, which has become a de facto standard in the Python dependency resolution literature. It has been used by both of our baselines [45,227], and contains realistic, challenging Python programs (Gists) exhibiting dependency drift, making it well suited for assessing module-level inference capabilities.

While `ReadPyE` has also been evaluated on other datasets, we focused on HG2.9K for two key reasons. First, it enables direct, fair comparison with prior work. Second, it allows us to isolate and evaluate the core question of this paper: how well can LLMs infer and resolve dependency configurations from Python source code? This controlled scope provides a focused evaluation of dependency inference capabilities, which is fundamental to broader dependency resolution challenges. Expanding to other datasets would improve the generalizability of our findings, and this is a natural next step for future work.

5.6.4. Conclusion validity

The HG2.9K dataset includes some Python programs that are unresolvable due to module deprecation. Additionally, our approach was validated only on x86 Docker Linux machines. This limitation restricts our ability to validate Gists in HG2.9K designed for other platforms, such as macOS or ARM-based systems. Hence, we were unable to validate if a Dockerfile generated by one of our baselines would execute correctly on these platforms.

As a result, `PyEGo` includes operating system-level dependencies by default. `ReadPyE` also provides this capability through a specific database dump. However, to keep the approaches consistent with `PLLM`, we configured `ReadPyE` to only fix Python-level dependencies with its default database.

[7] https://www.anthropic.com/claude/sonnet

5.7. Conclusion and Future Work

This paper introduced PLLM, a novel LLM-driven approach for resolving Python dependency conflicts. Unlike traditional methods such as PyEGo and ReadPyE, which rely on precomputed knowledge graphs, PLLM uses prompt-based inference and feedback loops to recover a working environment from raw Python code and runtime errors. This eliminates the need for extensive external infrastructure, making the solution lightweight, adaptable, and easy to apply across environments.

Our evaluation on the HG2.9K benchmark demonstrates that PLLM outperforms the state-of-the-art solutions PyEGo and ReadPyE. PLLM achieves significantly more successful resolutions, performing consistently, even in cases with many dependencies. Qualitative analysis reveals that PLLM's success stems from its ability to refine its guesses based on runtime information—an advantage absent in static systems.

At the same time, both baselines still succeed in a subset of cases that PLLM fails to resolve. These are often linked to legacy dependencies, renamed packages, or system-level nuances more effectively handled by structured knowledge graphs. This points to a key insight: prompt-based and knowledge-based techniques offer complementary strengths. Rather than treating them as competing approaches, we see clear potential in hybrid approaches that unify LLM-driven reasoning with curated symbolic knowledge.

Future work includes extending PLLM to support system-level dependency resolution, improving prompt construction and dependency disambiguation, and exploring alternative RAG methods such as GraphRAG [126]. Finally, we plan to investigate hybrid architectures that combine prompt-based inference with knowledge graph approaches given their complementary nature as highlighted by our results.

6

CONCLUSION

In 2022, we witnessed the wide-scale open adoption of Large Language Models (LLMs), marking a significant milestone in the field of natural language processing (NLP) and AI more broadly [143]. These models demonstrated unprecedented capabilities in generating human-like responses to a wide range of prompts, sparking widespread interest and excitement. Our methods of working evolved significantly during this period of AI growth, as our thesis has also enjoyed an evolutionary journey alongside the rapid advancements of AI in numerous aspects of our lives.

To more accurately reflect the validation of real-world AI-intensive systems, we implemented a black-box testing strategy, omitting the need for access to internal model states [209]. To effectively test these systems, we applied search-based software testing (SBST) techniques, which are well-suited for black-box testing scenarios [88]. As we progressed through our research, we observed that the complexity of the systems under test (SUT) required search-based methods to be augmented with more sophisticated techniques. This observation led us to develop and implement hybrid approaches that augment search-based methods with ML-based techniques. For example, in Chapter 3, we used a surrogate model as a proxy for the original SUT, allowing for more efficient evaluation of candidate test cases. By combining these techniques, we created novel, complementary approaches that leverage the strengths of SBST and ML-based techniques, resulting in more effective validation of AI-intensive systems.

In this final chapter, we synthesize the key contributions of this thesis by revisiting the four research questions (Section 1.3) and summarizing our empirical findings. We then outline the contributions to future impact of our work (Section 6.2), before finally outlining limitations and future research directions (Section 6.3) for testing AI intensive systems.

6.1. Research Questions Revisited

In this section, we revisit the four research questions in Section 1.3 and summarize the empirical findings from Chapters 2-5. Our findings validate the core hypothesis of this thesis (Chapter 1), while advancing the state-of-the-art in AI testing.

6.1.1. The Challenge of Minimal Perturbations in AI Testing

RQ1: *To what extent can search-based software techniques be applied to the generation of adversarial examples with minimal perturbation?*

The search space complexity (Section 1.2.3) of modern AI systems, particularly image classification models, poses a fundamental challenge for testing. While these models achieve remarkable accuracy under normal conditions, they remain vulnerable to adversarial examples: inputs with imperceptibly small perturbations that trigger incorrect predictions [77,200].

Traditional attack methods often rely on gradient-based approaches [77], which require thousands of iterations over the model. Moreover, many existing techniques generate non-minimal perturbations, by either applying uniforn noise or altering large regions of the input, resulting in unrealistic or easily detectable adversarial examples. Such methods are less effective for real-world applications (*e.g.,* autonomous driving), where such attacks fail to simulate plausible failure scenarios, limiting their utility for robust validation.

In Chapter 2, we addressed this gap by leveraging search-based software testing (SBST) to generate adversarial examples with minimal perturbations, while maintaining human interpretability. Our approach centered on Differential Evolution (DE) [146], a population-based optimization algorithm, that iteratively evolves candidate solutions towards a failure.

Unlike gradient-based methods, our approach avoids the need for model internals, making it a purely black-box technique. We focused on flipping the models's prediction with the smallest possible change to the input image. This was through our differential evolution algorithm, which optimizes for two objectives: maximizing the model's confidence in the incorrect class, while minimizing the number of pixels changed.

We evaluated our approach on five state-of-the-art Convolutional Neural Networks (CNNs) trained on the ImageNet dataset [58]. Using 150 randomly selected ground-truthed images from the ImageNet test set—a set not originally seen in the models training—we demonstrated that our DE-based method attained prediction flips with significantly fewer pixel changes than the baseline. Moreover, our method was significantly more efficient, requiring less iterations (calls to the model) compared to the baseline. Besides being more computational efficient, the reduced iterations required by our method also makes it applicable to real-world scenarios, as it is less likely to be detected by defenses that monitor for excessive model queries [153].

6.1.2. The critical role of diversity in AI testing

RQ2: *How does co-optimizing for diversity improve the effectiveness and generality of adversarial examples?*

While minimal perturbations (Section 6.1.1) ensure that adversarial examples are realistic, diversity ensures they are comprehensive—exposing a broad spectrum of vulnerabilities rather than isolated weaknesses.

This distinction is particularly critical for safety-critical systems, such as Deep Reinforcement Learning (DRL) models in autonomous driving, where undetected edge-case failures can have catastrophic consequences [133]. Traditional testing approaches often focus solely on failure detection, leading to repetitive or redundant test cases that miss rare but high-impact scenarios (*e.g.,* sharp turns, occlusions, or sensor failures).

In Chapter 3, we addressed this gap by incorporating diversity as a secondary objective in the generation of adversarial examples for DRL models. Our approach, `INDAGO-Nexus`, built upon the `INDAGO` framework [24], which focused solely on maximizing failure. We extended this framework with the addition of multi-objective evolutionary algorithms (MOEAs), specifically NSGA-II [57] and AGE-MOEA [161], to co-optimize for both failure and diversity.

We employed two diversity metrics: *Euclidean Distance* and *PCA-Based Clustering*, to understand how the calculation of diversity impacts the search process. Furthermore, we validated multiple pareto front points, including the extreme failure and knee point, to understand how the selection of test cases impacts the overall diversity of the selected tests.

Through this research, we found that co-optimizing for diversity led to `INDAGO-Nexus` uncovering significantly more unique failures than the baseline, particularly in the self-driving car (SDC) and parking scenarios. More so, our methods generated test cases more efficiently, achieving a 67% faster failure detection rate compared to the failure-only baseline.

In Chapter 4, we extended this focus on diversity to test case selection (TCS) and prioritization (TPS) for self-driving cars (SDCs), where executing all test cases is computationally prohibitive due to the sheer volume of potential scenarios.

To address this, we introduced a geometry-based image representation approach, where road scenarios are encoded as spatial images (*e.g.,* heatmaps of curvature, jerk). We then applied Convolutional Neural Networks (CNNs) to predict failure likelihoods using surrogate models trained on historical data. Using the surrogate for initial selection, we then applied a greedy algorithm to select diverse test cases from the selection pool. We evaluated our approach on a dataset of $\approx 16,000$ ground-truthed road scenarios.

By validating our approach against a number of baselines from the ISCT 2025 competition track [28], we found that our approach was effective at selecting diverse test cases, when compared against the state-of-the-art baselines.

6.1.3. Surrogate models in adversarial example generation and test case selection

RQ3: *How does the use of surrogate models impact the quality of test cases selected through search-based software testing techniques?*

As outlined above, we applied surrogate models to guide the search process for TCS and TPS for SDCs. The computational cost of testing AI-intensive systems, such as DRL models or large-scale simulators, presents a fundamental barrier to scalable and efficient validation (Section 1.2.2). Traditional SBST techniques, while effective, often require repeated, resource-intensive evaluations of the target model to converge on failure-inducing test cases. This challenges particularly problematic for DRL agents, where the search space is not only vast but also non-deterministic (Section 1.2.3), making exhaustive testing prohibitively expensive.

Surrogate models offer a promising solution to this challenge. By approximating the behavior of the target system, they enable the search process to be guided by a computationally efficient proxy, rather than the expensive target model. This approach reduces the need for repeated evaluations, making large-scale testing feasible while preserving the effectiveness of test case selection.

In Chapter 4, we investigated the impact of CNN based surrogate models on the quality of test cases selected for self-driving cars (SDCs). We trained a number of CNNs on historical simulation data to predict the likelihood of failure for given road scenarios. The idea of using a CNN as a surrogate model for road failure prediction, stems from the fact that road scenarios can be effectively represented as images. This is especially true, given the available data, which included a number of road points to plot the road with.

To understand the impact of surrogate quality, we evaluated multiple CNN architectures, performing hyperparameter tuning and ablation studies to ensure the robustness of each model before evaluating their impact on test case selection. Each model was then applied to the test case selection task, using our greedy selection approach and validated through the competition platform [28].

However, we also found that surrogate model quality and the selection threshold both had a significant impact on the quality of the selected test cases. In some models, a low threshold was necessary to select enough test cases, but this increased false positives; in others, a higher threshold was required, but this led to more false negatives.

6.1.4. THE REPRODUCIBILITY CHALLENGE IN AI TESTING

RQ4: *How can machine learning models help enable reproducible testing of AI-intensive systems?*

Before testing any system, we must first ensure it can run at all—a deceptively simple requirement, that becomes exceptionally complex for AI-intensive systems. The Oracle Problem (Section 1.2.1) becomes drastically more difficult, when the system under test is unable to run, or the libraries it depends on are different from the ones it was developed with.

This challenge is particularly poignant in Python-based AI workflows, where the rapid evolution of libraries (*e.g.,* Tensorflow, PyTorch, Numpy) combined with transitive dependencies, create a moving target for reproducibility. Traditional solutions, such as static knowledge graphs (*e.g.,* PyEGo [227] and ReadPyE [45]), require manual updates and struggle with the combinatorial explosion of version conflicts, particularly in ML based dependencies.

We hypothesized that Large Language Models (LLMs), with their contextual understanding of code, software projects and Natural Language Processing (NLP) capabilities, could directly infer the correct dependency versions for a given project. LLMs learn semantic relationships between dependencies (*e.g., scikit-learn* vs. *sklearn* naming conventions) and their versions from millions of Python projects. Unlike static methods, LLMs could also interpret runtime errors (*e.g., ImportError*, *ModuleNotFoundError*) to iteratively refine dependency suggestions.

Therefore, in Chapter 5, we presented a novel approach, called Python LLM (`PLLM`—pronounced 'plum') to directly answer our fourth thesis research question. This project emerged during an exciting time for the software engineering community, as the use of LLMs exponentially increased in both industry and academia [100,148] (Figure 6.1).

However, our initial experiments revealed a critical limitation: LLMs alone struggled with version hallucinations—suggesting plausible, but incorrect dependency combinations. This aligns with broader observations in the field [177], where models generate syntactically correct but semantically flawed outputs when lacking grounded context. To address this, `PLLM` implements a Retrieval Augmented Generation (RAG)-enhanced pipeline. It uses static analysis to extract import statements from source code, PyPI metadata retrieval to gather up-to-date dependency version and release information, and Docker-based validation with iterative repair to test candidate fixes and refine suggestions based on execution errors.

We evaluated `PLLM` on the HG2.9Kbenchmark, a curated dataset of $2,891$ 'hard' Python Gists with known dependency conflicts [98]. Our results demonstrated that the addition of RAG significantly improved `PLLM`'s performance. The RAG-enhanced `PLLM` achieved an increase over our baseline methods, reolving 21.58 more conflicts

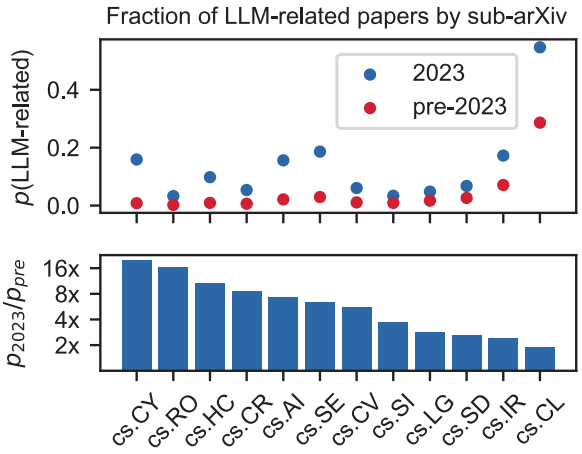


Figure 6.1. Sub-arXivs growing fastest in fraction of LLM papers [148].

that `PyEGo` and 15.97 more than `ReadPyE`. `PLLM` was particularly effective at resolving complex projects with multiple dependencies, where traditional static methods struggled.

6

While `PLLM` excelled at dynamic inference, the baselines succeeded in cases with legacy or renamed packages (*e.g., graphite* vs. *graphiti*), highlighting the complementary strenghts of static and dynamic approaches. This suggests that a hybrid approach, combining LLMs with static analysis, may be the most effective solution for dependency resolution in AI-intensive systems.

6.2. Contributions to Future Impact

In this section, we highlight several contributions from this thesis that are poised towards future impact extending beyond the publication of papers. Through these contributions, we aim to make our research accessible, while promoting adoption and expansion by the broader research and practitioner community:

Datasets: In Chapter 4, we contribute a newly curated dataset of $\approx 16,000$ ground-truthed test cases for SDCs. This dataset is important for the research community, as it provides a new set of ground-truthed data for training and evaluation. This dataset can be used to evaluate the robustness of self-driving car models, and to develop new approaches for test case selection. By providing the community with ground truth data, we increase the accessibility to clean, usable data.

Replication Packages: To ensure transparency and reproducibility, we provide complete replication packages for all empirical chapters.

Open Science: We have made every effort to ensure that our research is open and transparent. Therefore, we have used only open source tools and libraries, including

the models used in our research. Although this has possibly limited the performance of our approaches, we believe that it is important to ensure that our research is accessible to the widest possible audience. By refusing to use closed source models, such as GPT-4, we have ensured that our research can be reproduced and built upon by other researchers, without the need for access to proprietary models. This is particularly important in the context of AI intensive systems, where the use of closed source models can create barriers to entry for researchers who do not have access to these models.

`PLLM`: To further the research community in the area of dependency resolution, we have created a competition track, with `PLLM` as the baseline approach. The inaugural track "Agentic Based Python Dependency Resolution" [13] was held at the 2026 FSE-AIWare competition. The competition track is designed to encourage the development of new approaches for dependency resolution, and to provide a benchmark for evaluating these approaches. By providing a baseline approach, we hope to encourage researchers to develop new approaches that can outperform PLLM, and therefore further advance the state-of-the-art in dependency resolution.

6.3. Future Work

This thesis has highlighted several key challenges in the testing and validation of AI-intensive systems. Despite the novel approaches we have developed to address these challenges, significant opportunities remain for future research exploration.

6.3.1. Improved Search Strategies

Throughout this thesis, we have employed multiple search strategies to identify failures in AI-intensive systems, including Genetic Algorithms (GAs) and multi-objective evolutionary algorithms (MOEAs). We have adapted these algorithms through implementations of Differential Evolution (DE), custom crossover and mutation operators, and specialized selection strategies. However, considerable room for improvement remains in the search strategies used for testing AI-intensive systems. For example, the MIO algorithm [7] has demonstrated effectiveness for adversarial example generation in Deep Neural Networks (DNNs) [175], showing promise for further application to DRL agents. Tabu Search [74] has also proven effective for test case generation [62] and has been widely adopted in real-world applications such as scheduling and logistics [235]. That said, it is generally more difficult to implement than GAs due to the *memory* component, which requires managing a tabu list of previously visited solutions. Our work in Chapter 3 uses a stored list for calculating diversity, which could be further adapted to support a tabu list, opening promising new directions for testing and validating AI-intensive systems.

Additionally, the application of LLMs for search strategies represents an exciting area of emerging research. As LLMs have demonstrated increasing effectiveness and

growing adoption in recent years, we have observed substantial research implementing LLMs across diverse software testing tasks, including test case generation [66], test case prioritization [239], and fuzz testing [60]. However, significant opportunities for improvement remain in leveraging LLMs as search strategies. For instance, LLMs could generate novel candidate solutions or guide the search process by providing quality feedback on candidate solutions. This approach could prove particularly valuable for testing AI-intensive systems, where the search space is large and complex, and where conventional search strategies may struggle to identify failures.

6.3.2. Multi-agent testing

The scenarios we tested throughout this thesis were single-agent scenarios, where the system under test (SUT) was the only agent in the environment. While this approach is effective for validating individual agent robustness, it remains far from realistic. In real-world applications, DRL agents must interact with other agents, such as other vehicles or pedestrians. Multi-agent systems present several challenges, as they are inherently more complex and dynamic than their single-agent counterparts. Notably, high-dimensional seach spaces (Section 1.2.3) already pose a significant challenge when testing single-agent systems. The search space complexity increases exponentially in multi-agent systems, as we must account for the actions and interactions of multiple agents.

Furthermore, the incorporation of multi-agent systems exacerbates the simulation environment challenge (Section 1.2.2), requiring environments capable of supporting multi-agent interactions. While Gymnasium [36] is a popular environment, it offers relatively low fidelity and cannot fully capture the complexities of real-world multi-agent interactions [191]. Consequently, we would require high-fidelity simulators such as CARLA [65] or BeamNG [18], both of which support multi-agent interactions. However, these simulators are computationally expensive and demand dedicated hardware to execute effectively, constraining the number of test cases that can be performed. This computational bottleneck further limits researcher adoption and reduces research activity in this area. This research direction will likely intersect significantly with improved search strategies, as computational efficiency becomes paramount. Therefore, developing efficient testing methodologies for multi-agent systems remains a valuable future research direction.

6.3.3. Large scale ground-truthed datasets

Good datasets are crucial for the development and evaluation of new approaches for testing AI intensive systems. It was quality ground-truthed data, that allowed computer vision (CV) models to achieve their current level of performance [122]. The ILSVRC dataset [174], together with a competition platform, encouraged researchers to develop new approaches for image classification, and provided a benchmark for evaluating the approaches. Similarly, in the CV realm, the SnakeCLEF competition

[167,168], which is part of the larger LifeCLEF competition[1], provides numerous datasets for training and evaluating, together with a well-formulated competition platform.

These datasets and competition platforms have been crucial for the development of new approaches for creating and evaluating ML models [122], however, this remains lacking in the realm of autonomous vehicles. Although a number of competitions and datasets exist [23,212], there is still a need for more quality datasets and competition platforms for testing these systems. For example, the ICST 2025 competition track [28] has provided a comprehensive dataset of $\approx 30,000$ ground-truthed road scenarios. This dataset has been crucial in the development of new testing approaches for SDCs, and has provided a benchmark for evaluating these approaches.

Beyond this, when we consider the data used to train our surrogate models Chapter 3, we were able to utilise the data generated from the DRL agent's training. This data came from the `INDAGO` framework [24], in which the DRL agent was trained for the purpose of testing. However, there is likely a plethora of data being generated every day, from researchers and practitioners alike, that could be used to help train surrogate models. Therefore, we believe that future research, to explore methods of better collection and curation of this data, combined with competition platforms to encourage the use of, and generation of more data, would be a valuable contribution to the field. This would allow DRL testing approaches to hopefully reach the same notable heights as the ILSVRC competition did for CV.

6.3.4. Humans are Chaos Agents

It is hard to imagine a world where humans and autonomous vehicles exist on the same roads harmoniously [67]. Humans are unpredictable, and therefore, they are chaos agents. They can behave in ways that are unexpected and irrational, which can lead to accidents and other safety issues. Therefore, it is important to consider the impact of human behavior on the testing and validation of autonomous vehicles [109]. This could involve work similar to multi-agent testing, where the extra agents in the system are human-controlled, rather than AI-controlled and would allow us to better understand how our agents perform in more realistic scenarios, further improving the robustness of autonomous agents.

Beyond human behavior, it would be interesting to explore the impact of humans in the actual deployment of autonomous vehicles. For example, what if no humans were present on roads, and all vehicles were autonomous, working as a swarm to optimize traffic flow and safety [89,192]. This would be a drastic change from the current state of affairs, requiring a significant amount of research to understand the implications of such a change, and to develop the necessary infrastructure to support it. However, it could potentially lead to a safer and more efficient transportation

[1] https://www.imageclef.org/

system, as autonomous vehicles would be able to communicate with each other and coordinate their movements in ways that human drivers cannot.

BIBLIOGRAPHY

[1] Python software foundation - the pip tool, (2024).

[2] Building and distributing packages with setuptools, (2024).

[3] Z. Aghababaeyan, M. Abdellatif, L. Briand, R. S, M. Bagherzadeh, Black-box testing of deep neural networks through test case diversity, IEEE Trans. Softw. Eng. 49 (2023) 3182–3204.

[4] Z. Aghababaeyan, M. Abdellatif, L. Briand, R. S, M. Bagherzadeh, Black-box testing of deep neural networks through test case diversity, IEEE Transactions on Software Engineering 49 (2023) 3182–3204.

[5] Z. Aghababaeyan, M. Abdellatif, M. Dadkhah, L. Briand, DeepGD: A multi-objective black-box test selection approach for deep neural networks, ACM Trans. Softw. Eng. Methodol. 33 (2024).

[6] P. Altmeyer, A.M. Demetriou, A. Bartlett, C.C.S. Liem, Position: Stop making unscientific agi performance claims, in: Proceedings of the 41st International Conference on Machine Learning, JMLR.org, Vienna, Austria, 2024.

[7] A. Arcuri, Many independent objective (MIO) algorithm for test suite generation, in: T. Menzies, J. Petke (Eds.), Search Based Software Engineering, Springer International Publishing, Cham, 2017: pp. 3–17.

[8] B. Arya, J. Yao, J.L. Davy, M. Fard, A novel and fast crash simulation method: Revolutionising racetrack safety barrier analysis, Results in Engineering 25 (2025) 103870.

[9] P. Aryan, RoadGeom Transformer-SDCTest at the SBFT 2025 tool competition – CPS-SDC regression testing track, in: 2025 IEEE/ACM International Workshop on Search-Based and Fuzz Testing (SBFT), 2025: pp. 17–18.

[10] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, C. Sutton, Program synthesis with large language models, (2021).

[11] A. Bartlett, C.C.S. Liem, A. Panichella, On the strengths of pure evolutionary algorithms in generating adversarial examples, in: 2023 IEEE/ACM International Workshop on Search-Based and Fuzz Testing (SBFT), 2023: pp. 1–8.

[12] A. Bartlett, C.C.S. Liem, A. Panichella, Multi-objective differential evolution in the generation of adversarial examples, Science of Computer Programming 238 (2024) 103169.

[13] A. Bartlett, C.C.S. Liem, A. Panichella, FSE-Aiaware Tool competition 2026 – agentic based python dependency resolution, (2026).

[14] A. Bartlett, C. Liem, A. Panichella, The last dependency crusade: Solving python dependency conflicts with LLMs, in: 2025 40th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW), 2025: pp. 66–73.

[15] A. Bartlett, C. Liem, A. Panichella, The pursuit of diversity: Multi-objective testing of deep reinforcement learning agents, 16228 (2026).

[16] A. Bartlett, A. Panichella, Danger is my middle lane: Simulations from real-world dangerous roads, in: G. Jahangirova, F. Khomh (Eds.), Search-Based Software Engineering, Springer Nature Switzerland, Cham, 2024: pp. 77–83.

[17] A. Bartlett, A. Panichella, C. Liem, DRVN at the ICST 2025 tool competition – self-driving car testing track, in: 2025 IEEE Conference on Software Testing, Verification and Validation (ICST), 2025: pp. 807–808.

[18] BeamNG GmbH, BeamNG.tech, (2025).

[19] BeamNG GmbH, BeamNG.tech, (2025).

[20] R. Ben Abdessalem, A. Panichella, S. Nejati, L.C. Briand, T. Stifter, Testing autonomous cars for feature interaction failures using many-objective search, in: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, 2018: pp. 143–154.

[21] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, 13 (2012).

[22] A. Bhattacharyya, On a measure of divergence between two multinomial populations, Sankhyā: The Indian Journal of Statistics (1933-1960) 7 (1946) 401–406.

[23] M. Biagiola, S. Klikovits, SBFT tool competition 2024 - cyber-physical systems track, in: Proceedings of the 17th ACM/IEEE International Workshop on Search-Based and Fuzz Testing, Association for Computing Machinery, New York, NY, USA, 2024: pp. 33–36.

[24] M. Biagiola, P. Tonella, Testing of deep reinforcement learning agents with surrogate models, ACM Trans. Softw. Eng. Methodol. 33 (2024).

[25] C. Birchler, N. Ganz, Others, Cost-effective simulation-based test selection in self-driving cars software, Science of Computer Programming 226 (2023) 102926.

[26] C. Birchler, S. Khatiri, P. Derakhshanfar, S. Panichella, A. Panichella, Single and multi-objective test cases prioritization for self-driving cars in virtual environments, ACM Transactions on Software Engineering and Methodology 32 (2023) 1–30.

[27] C. Birchler, S. Khatiri, P. Rani, T. Kehrer, S. Panichella, A roadmap for simulation-based testing of autonomous cyber-physical systems: Challenges and future direction, ACM Trans. Softw. Eng. Methodol. 34 (2025).

[28] C. Birchler, S. Klikovits, M. Fazzini, S. Panichella, ICST tool competition 2025 - self-driving car testing track, in: 2025 IEEE Conference on Software Testing, Verification and Validation (ICST), IEEE Computer Society, Los Alamitos, CA, USA, 2025: pp. 801–804.

[29] C. Birchler, S. Klikovits, M. Fazzini, S. Panichella, SBFT tool competition 2025 - CPS-SDC regression testing track, in: 2025 IEEE/ACM International Workshop on Search-Based and Fuzz Testing (SBFT), IEEE Computer Society, Los Alamitos, CA, USA, 2025: pp. 3–4.

[30] C. Birchler, T.K. Mohammed, P. Rani, T. Nechita, T. Kehrer, S. Panichella, How does simulation-based testing for self-driving cars match human perception?, Proc. ACM Softw. Eng. 1 (2024).

[31] C. Birchler, C. Rohrbach, Others, SensoDat: Simulation-based sensor dataset of self-driving cars, in: Proceedings of the 21st International Conference on Mining Software Repositories, Association for Computing Machinery, New York, NY, USA, 2024: pp. 510–514.

[32] J. Blank, K. Deb, Pymoo: Multi-objective optimization in python, IEEE Access 8 (2020) 89497–89509.

[33] L. Bloch, A. Boketta, Others, Combination of object detection, geospatial data, and feature concatenation for snake species identification, in: Proceedings of the Working Notes of Clef 2022 - Conference and Labs of the Evaluation Forum, Bologna, Italy, September 5-8, 2022., CEUR-WS.org, 2022.

[34] J. Bloom, P. Paliwal, A. Mukherjee, C. Pinciroli, Decentralized multi-agent reinforcement learning with global state prediction, in: 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2023: pp. 8854–8861.

[35] B. Bracke, M. Bagherifar, Others, Joint feature learning of image data with embedded metadata to leverage snake species classification, in: Working Notes of the Conference and Labs of the Evaluation Forum (Clef 2023), Thessaloniki, Greece, 5-8 September, 2023., CEUR-WS.org, 2023.

[36] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, OpenAI gym, (2016).

[37] T. Brown, B. Mann, N. Ryder, M. Subbiah, J.D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, Language models are few-shot learners, in: H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, Curran Associates, Inc., 2020: pp. 1877–1901.

[38] S. Cass, The top programming languages 2024, IEEE Spectrum (2024).

[39] E. Castellano, A. Cetinkaya, Others, Frenetic at the SBST 2021 tool competition, in: 2021 IEEE/ACM 14th International Workshop on Search-Based Software Testing (SBST), 2021: pp. 36–37.

[40] E. Castellano, S. Klikovits, A. Cetinkaya, P. Arcaini, FreneticV at the SBST 2022 tool competition, in: Proceedings of the 15th Workshop on Search-Based Software Testing, Association for Computing Machinery, New York, NY, USA, 2023: pp. 47–48.

[41] K. Chan, B.H. Cheng, EvoAttack: An evolutionary search-based adversarial attack for object detection models, in: International Symposium on Search-Based Software Engineering, Springer, 2022: pp. 83–97.

[42] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. Pondé, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F.P. Such, D.W. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W.H. Guss, A. Nichol, I. Babuschkin, S. Balaji, S. Jain, A. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M.M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, W. Zaremba, Evaluating large language models trained on code, ArXiv abs/2107.03374 (2021).

[43] S. Chen, Y. Liao, F. Wang, G. Wang, L. Wang, Y. Wang, X. Zhu, Toward the robustness of autonomous vehicles in the AI era, The Innovation 6 (2025) 100780.

[44] Y. Chen, C. Ji, Y. Cai, T. Yan, B. Su, Deep reinforcement learning in autonomous car path planning and control: A survey, ArXiv abs/2404.00340 (2024).

[45] W. Cheng, W. Hu, X. Ma, Revisiting knowledge-based inference of python runtime environments: A realistic and adaptive approach, IEEE Transactions on Software Engineering 50 (2024) 258–279.

[46] F. Chollet, others, Keras, (2015).

[47] T. Chugh, K. Sindhya, J. Hakanen, K. Miettinen, A survey on handling computationally expensive multiobjective optimization problems with evolutionary algorithms, Soft Computing 23 (2019).

[48] M.B. Clowes, On seeing things, Artificial Intelligence 2 (1971) 79–116.

[49] W.J. Conover, Practical nonparametric statistics, John Wiley & Sons, 1998.

[50] W.J. Conover, Practical nonparametric statistics, john wiley & sons, 1999.

[51] J. Corona, R. Teixeira, M. Antunes, R. Aguiar, Intelligent task placement: Assessing the cost of deep reinforcement learning, in: 2025.

[52] M.J. Crawley, Statistics: An introduction using r, John Wiley & Sons, 2014.

[53] H. Cui, V. Radosavljevic, F.-C. Chou, T.-H. Lin, T. Nguyen, T.-K. Huang, J. Schneider, N. Djuric, Multimodal trajectory predictions for autonomous driving using deep convolutional networks, in: 2019 International Conference on Robotics and Automation (Icra), IEEE, 2019: pp. 2090–2096.

[54] I. Das, J.E. Dennis, Normal-boundary intersection: A new method for generating the pareto surface in nonlinear multicriteria optimization problems, SIAM Journal on Optimization 8 (1998) 631–657.

[55] K. Deb, Multi-objective optimisation using evolutionary algorithms: An introduction, in: Springer, 2011: pp. 3–34.

[56] K. Deb, H. Jain, An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints, IEEE Transactions on Evolutionary Computation 18 (2014) 577–601.

[57] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation 6 (2002) 182–197.

[58] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, ImageNet: A large-scale hierarchical image database, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition, 2009: pp. 248–255.

[59] J. Deng, J. Guo, N. Xue, S. Zafeiriou, ArcFace: Additive angular margin loss for deep face recognition, in: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019: pp. 4685–4694.

[60] Y. Deng, C.S. Xia, C. Yang, S.D. Zhang, S. Yang, L. Zhang, Large language models are edge-case generators: Crafting unusual programs for fuzzing deep learning libraries, in: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, Association for Computing Machinery, New York, NY, USA, 2024.

[61] X. Devroey, A. Gambi, J.P. Galeotti, R. Just, F. Kifetew, A. Panichella, S. Panichella, JUGE: An infrastructure for benchmarking java unit test generators, Software Testing, Verification and Reliability 33 (2023) e1838.

[62] E. Diaz, J. Tuya, R. Blanco, Automated software testing using a metaheuristic technique based on tabu search, in: 18th IEEE International Conference on Automated Software Engineering, 2003. Proceedings., 2003: pp. 310–313.

[63] J. Dietrich, D. Pearce, Others, Dependency versioning in the wild, in: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), 2019: pp. 349–359.

[64] C. Ding, X. He, K-means clustering via principal component analysis, in: Association for Computing Machinery, New York, NY, USA, 2004: p. 29.

[65] A. Dosovitskiy, G. Ros, Others, CARLA: An open urban driving simulator, (2017).

[66] K. El Haji, C. Brandt, A. Zaidman, Using github copilot for test generation in python: An empirical study, in: Proceedings of the 5th ACM/IEEE International Conference on Automation of Software Test (Ast 2024), Association for Computing Machinery, New York, NY, USA, 2024: pp. 45–55.

[67] B. Färber, Communication and communication problems between autonomous vehicles and human drivers, in: M. Maurer, J.C. Gerdes, B. Lenz, H. Winner (Eds.), Autonomous Driving, Springer, Berlin, Heidelberg, 2016.

[68] B. Fecher, S. Friesike, Open science: One term, five schools of thought, in: S. Bartling, S. Friesike (Eds.), Opening Science: The Evolving Guide on How the Internet Is Changing Research, Collaboration and Scholarly Publishing, Springer International Publishing, Cham, 2014: pp. 17–47.

[69] S. García, D. Molina, M. Lozano, F. Herrera, A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: A case study on the CEC'2005 special session on real parameter optimization, Journal of Heuristics 15 (2009) 617–644.

[70] Gemma Team, M. Riviere, S. Pathak, P.G. Sessa, C. Hardin, S. Bhupatiraju, L. Hussenot, T. Mesnard, B. Shahriari, A. Ramé, J. Ferret, P. Liu, P. Tafti, A. Friesen, M. Casbon, S. Ramos, R. Kumar, C.L. Lan, S. Jerome, A. Tsitsulin, N. Vieillard, P. Stanczyk, S. Girgin, N. Momchev, M. Hoffman, S. Thakoor, J.-B. Grill, B. Neyshabur, O. Bachem, A. Walton, A. Severyn, A. Parrish, A. Ahmad, A. Hutchison, A. Abdagic, A. Carl, A. Shen, A. Brock, A. Coenen, A. Laforge, A. Paterson, B. Bastian, B. Piot, B. Wu, B. Royal, C. Chen, C. Kumar, C. Perry, C. Welty, C.A. Choquette-Choo, D. Sinopalnikov, D. Weinberger, D. Vijaykumar, D. Rogozińska, D. Herbison, E. Bandy, E. Wang, E. Noland, E. Moreira, E. Senter, E. Eltyshev, F. Visin, G. Rasskin, G. Wei, G. Cameron, G. Martins, H. Hashemi, H. Klimczak-Plucińska, H. Batra, H. Dhand, I. Nardini, J. Mein, J. Zhou, J. Svensson, J. Stanway, J. Chan, J.P. Zhou, J. Carrasqueira, J. Iljazi, J. Becker, J. Fernandez, J. van Amersfoort, J. Gordon, J. Lipschultz, J. Newlan, J. Ji, K. Mohamed, K. Badola, K. Black, K. Millican, K. McDonell, K. Nguyen, K. Sodhia, K. Greene, L.L. Sjoesund, L. Usui, L. Sifre, L. Heuermann, L. Lago, L. McNealus, L.B. Soares, L. Kilpatrick, L. Dixon, L. Martins, M. Reid, M. Singh, M. Iverson, M. Görner, M. Velloso, M. Wirth, M. Davidow, M. Miller, M. Rahtz, M. Watson, M. Risdal, M. Kazemi, M. Moynihan, M. Zhang, M. Kahng, M. Park, M. Rahman, M. Khatwani, N. Dao, N. Bardoliwalla, N. Devanathan, N. Dumai, N. Chauhan, O. Wahltinez, P. Botarda, P. Barnes, P. Barham, P. Michel, P. Jin, P. Georgiev, P. Culliton, P. Kuppala, R. Comanescu, R. Merhej, R. Jana, R.A. Rokni, R. Agarwal, R. Mullins, S. Saadat, S.M. Carthy, S. Cogan, S. Perrin, S.M.R. Arnold, S. Krause, S. Dai, S. Garg, S. Sheth, S. Ronstrom, S. Chan, T. Jordan, T. Yu, T. Eccles, T. Hennigan, T. Kocisky, T. Doshi, V. Jain, V. Yadav, V. Meshram, V. Dharmadhikari, W. Barkley, W. Wei, W. Ye, W. Han, W. Kwon, X. Xu, Z. Shen, Z. Gong, Z. Wei, V. Cotruta, P. Kirk, A. Rao, M. Giang, L. Peran, T. Warkentin, E. Collins, J. Barral, Z. Ghahramani, R. Hadsell, D. Sculley, J. Banks, A. Dragan, S. Petrov, O. Vinyals, J. Dean, D. Hassabis, K. Kavukcuoglu, C. Farabet, E. Buchatskaya, S. Borgeaud, N. Fiedel, A. Joulin, K. Kenealy, R. Dadashi, A. Andreev, Gemma 2: Improving open language models at a practical size, (2024).

[71] S. Ghamizi, M. Cordy, M. Gubri, M. Papadakis, A. Boystov, Y. Le Traon, A. Goujon, Search-based adversarial testing and improvement of constrained credit scoring systems, in: Proceedings of the 28th ACM Esec/Fse, Association for Computing Machinery, New York, NY, USA, 2020: pp. 1089–1100.

[72] A. Ghoshal, Why did this humanoid robot go nuts and nearly injure its handlers?, New Atlus (2025).

[73] R. Giubilato, A. Masili, S. Chiodini, M. Pertile, S. Debei, Simulation framework for mobile robots in planetary-like environments, in: 2020 IEEE 7th International Workshop on Metrology for AeroSpace (MetroAeroSpace), 2020: pp. 594–599.

[74] F. Glover, R. Marti, Tabu search, in: E. Alba, R. Martí (Eds.), Metaheuristic Procedures for Training Neutral Networks, Springer US, Boston, MA, 2006: pp. 53–69.

[75] D.E. Goldberg, J.H. Holland, Genetic algorithms and machine learning, Machine Learning 3 (1988) 95–99.

[76] I. Goodfellow, Y. Bengio, A. Courville, Deep learning, MIT Press, 2016.

[77] I.J. Goodfellow, J. Shlens, C. Szegedy, Explaining and harnessing adversarial examples, in: Proceedings of the International Conference on Learning Representations (Iclr), 2015.

[78] Google AI Blog, Reducing the computational cost of deep reinforcement learning research, (2021).

[79] A. Grattafiori, A. Dubey, Others, The llama 3 herd of models, (2024).

[80] T. Guardian, Us investigates 2.4m tesla self-driving vehicles after reported collisions, The Guardian (2024).

[81] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, D. Pedreschi, A survey of methods for explaining black box models, ACM Computing Surveys (CSUR) 51 (2018) 1–42.

[82] A. Güllü, F.A. Shah, D. Pfahl, ITS4SDC at the ICST 2025 tool competition - self-driving car testing track, in: 2025 IEEE Conference on Software Testing, Verification and Validation (ICST), 2025: pp. 809–810.

[83] D. Guo, D. Yang, H. Zhang, others, DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning, Nature 645 (2025) 633–638.

[84] J. Guo, Y. Zhao, H. Song, Y. Jiang, Coverage guided differential adversarial testing of deep learning systems, IEEE Transactions on Network Science and Engineering 8 (2021) 933–942.

[85] I. Habli, T. Lawton, Z. Porter, Artificial intelligence in health care: Accountability and safety, Bulletin of the World Health Organization 98 (2020) 251–256.

[86] F.U. Haq, D. Shin, L. Briand, Efficient online testing for DNN-enabled systems using surrogate-assisted and many-objective optimization, in: 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE), 2022: pp. 811–822.

[87] M. Harman, B.F. Jones, Search-based software engineering, Information and Software Technology 43 (2001) 833–839.

[88] M. Harman, S.A. Mansouri, Y. Zhang, Search-based software engineering: Trends, techniques and applications, ACM Comput. Surv. 45 (2012).

[89] A. Hashim, T. Saini, H. Bhardwaj, A. Jothi, A. Vinay Kumar, Application of swarm intelligence in autonomous cars for obstacle avoidance, in: A. Krishna, K. Srikantaiah, C. Naveena (Eds.), Integrated Intelligent Computing, Communication and Security, Springer, Singapore, 2019.

[90] N. Hatami, Y. Gavet, J. Debayle, Classification of time-series images using deep convolutional neural networks, in: Proceedings of the 10th International Conference on Machine Vision (Icmv 2017), SPIE, 2018: p. 106960Y.

[91] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, 2016.

[92] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016: pp. 770–778.

[93] C. Hensley, M. Marshall, Off-road navigation with end-to-end imitation learning for continuously parameterized control*, SoutheastCon 2022 (2022) 591–597.

[94] M.C. Hinojosa Lee, J. Braet, J. Springael, Performance metrics for multilabel emotion classification: Comparing micro, macro, and weighted F1-scores, Applied Sciences 14 (2024) 9863.

[95] G.E. Hinton, O. Vinyals, J. Dean, Distilling the knowledge in a neural network, ArXiv abs/1503.02531 (2015).

[96] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Computation 9 (1997) 1735–1780.

[97] J.H. Holland, Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence, MIT press, 1992.

[98] E. Horton, C. Parnin, Gistable: Evaluating the executability of python code snippets on github, in: 2018: pp. 217–227.

[99] E. Horton, C. Parnin, DockerizeMe: Automatic inference of environment dependencies for python code snippets, (2019).

[100] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, H. Wang, Large language models for software engineering: A systematic literature review, ACM Trans. Softw. Eng. Methodol. 33 (2024).

[101] F. Hu, P. Wang, Others, Watch out venomous snake species: A solution to Snakeclef2023, in: Working Notes of the Conference and Labs of the Evaluation Forum (Clef 2023), Thessaloniki, Greece, 5-8 September, 2023., CEUR-WS.org, 2023.

[102] K. Huang, B. Chen, Others, Demystifying dependency bugs in deep learning stack, in: Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, New York, NY, USA, 2023: pp. 450–462.

[103] D. Huffman, Impossible objects as nonsense sentences, 6 (1971).

[104] D. Humeniuk, F. Khomh, G. Antoniol, AmbieGen: A search-based framework for autonomous systems testing, Sci. Comput. Program. 230 (2023).

[105] H. Hüttel, On program synthesis and large language models, Commun. ACM 68 (2024) 33–35.

[106] H. Ishibuchi, N. Tsukamoto, Y. Nojima, Evolutionary many-objective optimization: A short review, in: IEEE, 2008: pp. 2419–2426.

[107] S. Ishida, S. Ono, Adjust-free adversarial example generation in speech recognition using evolutionary multi-objective optimization under black-box condition, Artif. Life Robot. 26 (2021) 243–249.

[108] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, D. Kalenichenko, Quantization and training of neural networks for efficient integer-arithmetic-only inference, in: 2018: pp. 2704–2713.

[109] B. Jafary, E. Rabiei, M.A. Diaconeasa, H. Masoomi, L. Fiondella, A. Mosleh, A survey on autonomous vehicles interactions with human and other vehicles, in: 14th Psam International Conference on Probabilistic Safety Assessment and Management, IAPSAM Los Angeles, 2018.

[110] A.Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D.S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L.R. Lavaud, M.-A. Lachaux, P. Stock, T.L. Scao, T. Lavril, T. Wang, T. Lacroix, W.E. Sayed, Mistral 7b, (2023).

[111] A. Juels, M. Wattenberg, Stochastic hillclimbing as a baseline method for evaluating genetic algorithms, in: D. Touretzky, M.C. Mozer, M. Hasselmo (Eds.), Advances in Neural Information Processing Systems, MIT Press, 1995.

[112] M. Kaufeld, K. Moller, A. Gambi, P. Arcaini, J. Betz, MultiDrive: A co-simulation framework bridging 2D and 3D driving simulation for AV software validation, in: 2025 IEEE 28th International Conference on Intelligent Transportation Systems (ITSC), 2025: pp. 1596–1602.

[113] P. Kaur, S. Taghavi, Z. Tian, W. Shi, A survey on simulators for testing self-driving cars, in: 2021 Fourth International Conference on Connected and Autonomous Driving (MetroCAD), 2021: pp. 62–70.

[114] K. Khandelwal, A study to know - use of AI for personalized recommendation, streaming optimization, and original content production at netflix, International Journal of Scientific Research and Engineering Trends 9 (2023) 1738–1743.

[115] S. Khare, R. Aralikatte, S. Man, Adversarial black-box attacks on automatic speech recognition systems using multi-objective evolutionary optimization, in: Proceedings of Interspeech, 2019.

[116] F.M. Kifetew, A. Panichella, A. De Lucia, R. Oliveto, P. Tonella, Orthogonal exploration of the search space in evolutionary test case generation, in: Proceedings of the 2013 International Symposium on Software Testing and Analysis, Association for Computing Machinery, New York, NY, USA, 2013: pp. 257–267.

[117] J. Kim, R. Feldt, S. Yoo, Guiding deep learning system testing using surprise adequacy, in: Proceedings of the 41st International Conference on Software Engineering, IEEE Press, Montreal, Quebec, Canada, 2019: pp. 1039–1049.

[118] B.R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A.A.A. Sallab, S. Yogamani, P. Pérez, Deep reinforcement learning for autonomous driving: A survey, IEEE Transactions on Intelligent Transportation Systems 23 (2022) 4909–4926.

[119] S. Klikovits, V. Riccio, E. Castellano, A. Cetinkaya, A. Gambi, P. Arcaini, Does road diversity really matter in testing automated driving systems? - a registered report, ArXiv abs/2209.05947 (2022).

[120] T. Kramer, contributors, Self driving car sandbox, (2021).

[121] A. Krizhevsky, Learning multiple layers of features from tiny images, 2009.

[122] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: F. Pereira, C.J. Burges, L. Bottou, K. Weinberger (Eds.), Advances in Neural Information Processing Systems, Curran Associates, Inc., 2012.

[123] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Commun. ACM 60 (2017) 84–90.

[124] F. Lambert, Tesla full self-driving veers off road, flips car in scary crash driver couldn't prevent, Electrek (2025).

[125] M. Landers, A. Doryab, Deep reinforcement learning verification: A survey, ACM Comput. Surv. 55 (2023).

[126] J. Larson, S. Truitt, GraphRAG: Unlocking LLM discovery on narrative private data, Microsoft Research Blog (2024).

[127] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, L.D. Jackel, Backpropagation applied to handwritten zip code recognition, Neural Computation 1 (1989) 541–551.

[128] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (1998) 2278–2324.

[129] E. Leurent, An environment for autonomous driving decision-making, GitHub Repository (2018).

[130] P. Li, T. Sun, Q. Tang, H. Yan, Y. Wu, X. Huang, X. Qiu, CodeIE: Large code generation models are better few-shot information extractors, in: A. Rogers, J. Boyd-Graber, N. Okazaki (Eds.), Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Toronto, Canada, 2023: pp. 15339–15353.

[131] C.C.S. Liem, A. Panichella, Oracle issues in machine learning and where to find them, in: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, 2020.

[132] J. Lin, L. Xu, Y. Liu, X. Zhang, Black-box adversarial sample generation based on differential evolution, Journal of Systems and Software 170 (2020) 110767.

[133] H.X. Liu, S. Feng, Curse of rarity for autonomous vehicles, Nature Communications 15 (2024) 4808.

[134] P.A. Lopez, M. Behrisch, Others, Microscopic traffic simulation using sumo, in: The 21st IEEE International Conference on Intelligent Transportation Systems, IEEE, 2018.

[135] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao, Y. Wang, DeepMutation: Mutation testing of deep learning systems, in: Proceedings of the 29th International Symposium on Software Reliability Engineering (Issre), 2018.

[136] Y. Man, R. Muller, M. Li, Z.B. Celik, R. Gerdes, That person moves like a car: Misclassification attack detection for autonomous systems using spatiotemporal consistency, in: Proceedings of the 32nd Usenix Conference on Security Symposium, USENIX Association, USA, 2023.

[137] H.B. Mann, D.R. Whitney, On a test of whether one of two random variables is stochastically larger than the other, Annals of Mathematical Statistics 18 (1947) 50–60.

[138] A. Marinov, P. Arcaini, A. Bartlett, A. Gambi, F. Ishikawa, A. Panichella, Efficient exploration of autonomous driving system safety boundaries, in: 2026 IEEE Intelligent Vehicles Symposium (IV), 2026: pp. 1261–1266.

[139] S. Messaoudi, A. Panichella, Others, A search-based approach for accurate identification of log message formats, in: 2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC), 2018: pp. 167–16710.

[140] I.D. Mienye, T.G. Swart, G. Obaido, M. Jordan, P. Ilono, Deep convolutional neural networks in medical image analysis: A review, Information 16 (2025) 195.

[141] I.D. Mienye, T.G. Swart, G. Obaido, M. Jordan, P. Ilono, Deep convolutional neural networks in medical image analysis: A review, Information 16 (2025).

[142] E.F. Miller, R.A. Melton, Automated generation of testcase datasets, in: Proceedings of the International Conference on Reliable Software, Association for Computing Machinery, New York, NY, USA, 1975: pp. 51–58.

[143] S. Minaee, T. Mikolov, N. Nikzad, M.A. Chenaghlu, R. Socher, X. Amatriain, J. Gao, Large language models: A survey, ArXiv abs/2402.06196 (2024).

[144] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning, (2013).

[145] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M.A. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature 518 (2015) 529–533.

[146] J. Montgomery, S. Chen, An analysis of the operation of differential evolution at high and low crossover rates, in: IEEE Congress on Evolutionary Computation, IEEE, 2010: pp. 1–8.

[147] S.-M. Moosavi-Dezfooli, A. Fawzi, P. Frossard, DeepFool: A simple and accurate method to fool deep neural networks, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016: pp. 2574–2582.

[148] R. Movva, S. Balachandar, K. Peng, G. Agostini, N. Garg, E. Pierson, Topics, authors, and institutions in large language model research: Trends from 17K arXiv papers, in: K. Duh, H. Gomez, S. Bethard (Eds.), Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), Association for Computational Linguistics, Mexico City, Mexico, 2024: pp. 1223–1243.

[149] R. Mukherjee, K.S. Patnaik, A survey on different approaches for software test case prioritization, Journal of King Saud University - Computer and Information Sciences 33 (2021) 1041–1054.

[150] S. Mukherjee, A. Almanza, C. Rubio-González, Fixing dependency errors for python build reproducibility, in: Association for Computing Machinery, New York, NY, USA, 2021: pp. 439–451.

[151] N. Müller, T. Glasmachers, Challenges in high-dimensional reinforcement learning with evolution strategies, in: A. Auger, C. Fonseca, N. Lourenço, P. Machado, L. Paquete, D. Whitley (Eds.), Parallel Problem Solving from Nature – Ppsn Xv, Springer, Cham, 2018.

[152] S. Muresan, E. Tzoukermann, J.L. Klavans, Combining linguistic and machine learning techniques for email summarization, in: Proceedings of the ACL 2001 Workshop on Computational Natural Language Learning (CONLL), 2001.

[153] R. Muthalagu, J. Malik, P.M. Pawar, Detection and prevention of evasion attacks on machine learning models, Expert Systems with Applications 266 (2025) 126044.

[154] P. Nemenyi, Distribution-free multiple comparisons, PhD Thesis, Princeton University (1963).

[155] A. Nguyen, J. Yosinski, J. Clune, Deep neural networks are easily fooled: High confidence predictions for unrecognizable images, in: Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (Cvpr), 2015.

[156] C. Nie, H. Leung, A survey of combinatorial testing, ACM Comput. Surv. 43 (2011).

[157] B.A. Nosek, G. Alter, G.C. Banks, D. Borsboom, S.D. Bowman, S.J. Breckler, S. Buck, C.D. Chambers, G. Chin, G. Christensen, M. Contestabile, A. Dafoe, E. Eich, J. Freese, R. Glennerster, D. Goroff, D.P. Green, B. Hesse, M. Humphreys, J. Ishiyama, D. Karlan, A. Kraut, A. Lupia, P. Mabry, T. Madon, N. Malhotra, E. Mayo-Wilson, M. McNutt, E. Miguel, E.L. Paluck, U. Simonsohn, C. Soderberg, B.A. Spellman, J. Turitto, G. VandenBos, S. Vazire, E.J. Wagenmakers, R. Wilson, T. Yarkoni, Promoting an open research culture, Science 348 (2015) 1422–1425.

[158] OpenAI Team, GPT-4 technical report, (2024).

[159] B. Paden, M. Čáp, Others, A survey of motion planning and control techniques for self-driving urban vehicles, IEEE Transactions on Intelligent Vehicles 1 (2016) 33–55.

[160] A. Panichella, An adaptive evolutionary algorithm based on non-euclidean geometry for many-objective optimization, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2019: pp. 595–603.

[161] A. Panichella, An adaptive evolutionary algorithm based on non-euclidean geometry for many-objective optimization, in: Association for Computing Machinery, New York, NY, USA, 2019: pp. 595–603.

[162] A. Panichella, A systematic comparison of search-based approaches for lda hyperparameter tuning, Information and Software Technology 130 (2021) 106411.

[163] A. Panichella, C.C.S. Liem, What are we really testing in mutation testing for machine learning? A critical reflection, in: Proceedings of the IEEE/ACM 43rd International Conference on Software Engineering - Nier Track, 2021.

[164] S. Panichella, P. Arcaini, M.B. Cohen, A. Arrieta, eds., Roadmap for DevOps in Cyber-Physical Systems: Challenges and future directions, Springer Nature Singapore, Singapore, 2026.

[165] K. Pei, Y. Cao, J. Yang, S. Jana, DeepXplore: Automated whitebox testing of deep learning systems, Commun. ACM 62 (2019) 137–145.

[166] Ó. Pérez-Gil, R. Barea, E. López-Guillén, L.M. Bergasa, C. Gómez-Huélamo, R. Gutiérrez, A. Díaz-Díaz, Deep reinforcement learning based control for autonomous vehicles in CARLA, Multimed Tools Appl 81 (2022) 3553–3576.

[167] L. Picek, M. Hrúz, A.M. Durso, Overview of SnakeCLEF 2024: Revisiting snake species identification in medically important scenarios, in: Working Notes of the Conference and Labs of the Evaluation Forum (Clef 2024), Grenoble, France, 9-12 September, 2024., CEUR-WS.org, 2024: p. 188.

[168] L. Picek, M. Hrúz, Others, Overview of SnakeCLEF 2022: Automated snake species identification on a global scale, in: Proceedings of the Working Notes of CLEF 2022 - Conference and Labs of the Evaluation Forum, Bologna, Italy, September 5-8, 2022., CEUR-WS.org, 2022.

[169] R. Rajamani, Vehicle dynamics and control, Springer US, 2011.

[170] L. Roberts, Machine perception of three-dimensional solids, 1963.

[171] L.G. Roberts, Machine perception of 3-d solids, in: Optical and Electro-Optical Information Processing, MIT Press, 1965.

[172] N.A. Rosy, K. Balasubadra, K. Deepa, Are vision transformers replacing convolutional neural networks in scene interpretation?: A review, Discover Applied Sciences 7 (2025) 932.

[173] A. Rovetta, Raiders of the lost correlation: A guide on using pearson and spearman coefficients to detect hidden correlations in medical sciences, Cureus 12 (2020) e11794.

[174] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A.C. Berg, L. Fei-Fei, Imagenet large scale visual recognition challenge, Int. J. Computer Vision 115 (2015) 211–252.

[175] O. Sahin, Generation of black-box adversarial attacks using many independent objective-based algorithm for testing the robustness of deep neural networks, Appl. Soft Comput. 164 (2024).

[176] T. Salimans, J. Ho, X. Chen, S. Sidor, I. Sutskever, Evolution strategies as a scalable alternative to reinforcement learning, arXiv Preprint arXiv:1703.03864 (2017).

[177] J. Sallou, T. Durieux, A. Panichella, Breaking the silence: The threats of using LLMs in software engineering, in: Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results, Association for Computing Machinery, New York, NY, USA, 2024: pp. 102–106.

[178] J. Schmidhuber, Deep learning in neural networks: An overview, Neural Networks 61 (2015) 85–117.

[179] H.-P. Schoener, The role of simulation in development and testing of autonomous vehicles, in: 2017.

[180] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, CoRR abs/1707.06347 (2017).

[181] B. Selman, C.P. Gomes, Hill-climbing search, in: Encyclopedia of Cognitive Science, John Wiley & Sons, Ltd, 2006.

[182] K. Simonyan, A. Vedaldi, A. Zisserman, Deep inside convolutional networks: Visualising image classification models and saliency maps, (2014).

[183] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, CoRR abs/1409.1556 (2014).

[184] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, in: Proceedings of the 3rd International Conference on Learning Representations (ICLR), 2015.

[185] A.W.M. Smeulders, M. Worring, S. Santini, A. Gupta, R. Jain, Content-based image retrieval at the end of the early years, IEEE Trans. Pattern Analysis and Machine Intelligence 22 (2000) 1349–1380.

[186] K.W. Smith, Cython: A guide for python programmers, " O'Reilly Media, Inc.", 2015.

[187] M. Srinivas, L.M. Patnaik, Adaptive probabilities of crossover and mutation in genetic algorithms, IEEE Transactions on Systems, Man, and Cybernetics 24 (1994) 656–667.

[188] S. Stabinger, A. Rodríguez-Sánchez, J. Piater, 25 years of CNNs: Can we compare to human abstraction capabilities?, in: A. Villa, P. Masulli, A. Pons Rivero (Eds.), Artificial Neural Networks and Machine Learning – Icann 2016, Springer, Cham, 2016.

[189] Stanford Institute for Human-Centered Artificial Intelligence (HAI), The 2025 AI index report, Stanford University, 2025.

[190] L. St»hle, S. Wold, Analysis of variance (ANOVA), Chemometrics and Intelligent Laboratory Systems 6 (1989) 259–272.

[191] A. Stocco, B. Pulfer, P. Tonella, Mind the gap! A study on the transferability of virtual versus physical-world testing of autonomous driving systems, IEEE Transactions on Software Engineering 49 (2023) 1928–1940.

[192] D. St-Onge, M. Kaufmann, J. Panerati, B. Ramtoula, Y. Cao, E.B.J. Coffey, G. Beltrame, Planetary exploration with robot teams: Implementing higher autonomy with swarm intelligence, IEEE Robotics & Automation Magazine 27 (2020) 159–168.

[193] R. Storn, K. Price, Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces, J. Of Global Optimization 11 (1997) 341–359.

[194] J. Su, D.V. Vargas, K. Sakurai, One pixel attack for fooling deep neural networks, IEEE Transactions on Evolutionary Computation 23 (2019) 828–841.

[195] B. Sun, S. Mohamed Haris, R. Ramli, A systematic review of deep reinforcement learning for legged robot locomotion, Instruments 10 (2026) 8.

[196] W. Sun, J. Jin, W. Lin, Minimum noticeable difference-based adversarial privacy preserving image generation, IEEE Trans. Cir. And Sys. For Video Technol. 33 (2023) 1069–1081.

[197] R.S. Sutton, A.G. Barto, Reinforcement learning: An introduction, Second, The MIT Press, 2018.

[198] T. Suzuki, S. Takeshita, S. Ono, Adversarial example generation using evolutionary multi-objective optimization, in: 2019 IEEE Congress on Evolutionary Computation (Cec), 2019: pp. 2136–2144.

[199] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016: pp. 2818–2826.

[200] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus, Intriguing properties of neural networks, in: Proceedings of the International Conference on Learning Representations (Iclr), 2014.

[201] H. Takita, D. Kabata, S. Walston, H. Tatekawa, K. Saito, Y. Tsujimoto, Y. Miki, D. Ueda, A systematic review and meta-analysis of diagnostic performance comparison between generative AI and physicians, Npj Digital Medicine 8 (2025).

[202] A. Tammewar, N. Chaudhari, B. Saini, D. Venkatesh, G. Dharahas, D. Vora, S. Patil, K. Kotecha, S. Alfarhood, Improving the performance of autonomous driving through deep reinforcement learning, Sustainability 15 (2023) 13799.

[203] M. Tan, Q.V. Le, EfficientNet: Rethinking model scaling for convolutional neural networks, (2020).

[204] C. Tang, B. Abbatematteo, J. Hu, R. Chandra, R. Martín-Martín, P. Stone, Deep reinforcement learning for robotics: A survey of real-world successes, Annual Review of Control, Robotics, and Autonomous Systems 8 (2025) 153–188.

[205] Y. Tian, K. Pei, S. Jana, B. Ray, DeepTest: Automated testing of deep-neural-network-driven autonomous cars, in: Proceedings of the 40th International Conference on Software Engineering, Association for Computing Machinery, New York, NY, USA, 2018: pp. 303–314.

[206] E. Todorov, T. Erez, Y. Tassa, MuJoCo: A physics engine for model-based control, 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (2012) 5026–5033.

[207] M.F. Triola, Elementary statistics, Pearson/Addison-Wesley Reading, MA, 2004.

[208] B. Udugama, Review of deep reinforcement learning for autonomous driving, (2023).

[209] J. Uesato, A. Kumar, C. Szepesvari, T. Erez, A. Ruderman, K. Anderson, Krishmamurthy, Dvijotham, N. Heess, P. Kohli, Rigorous agent evaluation: An adversarial approach to uncover catastrophic failures, (2018).

[210] S. Ullah, M. Masood, Genetic drift and its effects on the performance of genetic algorithm(GA), in: 2023 International Conference on Robotics and Automation in Industry (Icrai), 2023: pp. 1–5.

[211] United Nations Conference on Trade and Development (UNCTAD), Issues paper on science, technology and innovation in the age of AI, United Nations, 2025.

[212] R.E. Uysal, A. Javadi, P. Aryan, A. Babikian, D. Humeniuk, S. Mazraehkhatiri, S. Panichella, SBFT tool competition 2026 – UAV testing track, in: International Workshop on Search-Based and Fuzz Testing, Sbft@icse 2026, 2026.

[213] A. Vargha, H.D. Delaney, A critique and improvement of the cl common language effect size statistics of McGraw and Wong, Journal of Educational and Behavioral Statistics 25 (2000) 101–132.

[214] K.R. Varshney, Engineering safety in machine learning, in: 2016 Information Theory and Applications Workshop (Ita), IEEE, 2016.

[215] D.L. Waltz, Understanding line drawings of scenes with shadows, in: 1975.

[216] F. Wang, Z. Zhang, X. Zhang, Z. Wu, T. Mo, Q. Lu, W. Wang, R. Li, J. Xu, X. Tang, Q. He, Y. Ma, M. Huang, S. Wang, A comprehensive survey of small language models in the era of large language models: Techniques, enhancements, applications, collaboration with llms, and trustworthiness, ACM Trans. Intell. Syst. Technol. 16 (2025).

[217] H. Wang, J.K. Kearney, K. Atkinson, K. Atkinson, Arc-length parameterized spline curves for real-time simulation, in: 2003.

[218] Y. Wang, S. Xing, C. Can, R. Li, H. Hua, K. Tian, Z. Mo, X. Gao, K. Wu, S. Zhou, H. You, J. Peng, J. Zhang, Z. Wang, R. Song, M. Yan, W. Zimmer, X. Zhou, P. Li, Z. Lu, C.-J. Chen, Y. Huang, R.A. Rossi, L. Sun, H. Yu, Z. Fan, F.H. Yang, Y. Kang, R. Greer, C. Liu, E.H. Lee, X. Di, X. Ye, L. Ren, A. Knoll, X. Li, S. Ji, M. Tomizuka, M. Pavone, T. Yang, J. Du, M.-H. Yang, H. Wei, Z. Wang, Y. Zhou, J. Li, Z. Tu, Generative AI for autonomous driving: Frontiers and opportunities, (2025).

[219] Z. Wang, H. Yan, C. Wei, J. Wang, S. Bo, M. Xiao, Research on autonomous driving decision-making strategies based deep reinforcement learning, in: Proceedings of the 2024 4th International Conference on Internet of Things and Machine Learning, Association for Computing Machinery, New York, NY, USA, 2024: pp. 211–215.

[220] A.L. Watkins, The automatic generation of test data using genetic algorithms, in: Conference Proceedings, Dundee, UK, 1995.

[221] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E.H. Chi, Q.V. Le, D. Zhou, Chain-of-thought prompting elicits reasoning in large language models, in: Curran Associates Inc., Red Hook, NY, USA, 2022.

[222] P.H. Winston, The psychology of computer vision, Pattern Recognition 8 (1976) 193.

[223] M. Woelfle, P. Olliaro, M. Todd, Open science is a research accelerator, Nature Chemistry 3 (2011) 745–748.

[224] S. Woo, S. Debnath, Others, ConvNeXt V2: Co-designing and scaling convnets with masked autoencoders, (2023).

[225] S. Xanthakis, C. Ellis, C. Skourlas, A. Le Gall, S. Katsikas, Application of genetic algorithms to software testing, in: Proceedings of the 5th International Conference on Software Engineering, Toulouse, France, 1992.

[226] D. Yang, A. Hussain, C.V. Lopes, From query to usable code: An analysis of stack overflow code snippets, in: Proceedings of the 13th International Conference on Mining Software Repositories, Association for Computing Machinery, New York, NY, USA, 2016: pp. 391–402.

[227] H. Ye, W. Chen, Others, Knowledge-based environment dependency inference for python programs, in: Association for Computing Machinery, New York, NY, USA, 2022: pp. 1245–1256.

[228] S. Yoo, SBST in the age of machine learning systems - challenges ahead, in: 2019 IEEE/ACM 12th International Workshop on Search-Based Software Testing (SBST), 2019: pp. 2–2.

[229] S. Yoo, M. Harman, Regression testing minimization, selection and prioritization: A survey, Software Testing, Verification and Reliability 22 (2012) 67–120.

[230] J. Zhang, Z. Cui, X. Chen, H. Yang, L. Zheng, J. Liu, CIDFuzz: Fuzz testing for continuous integration, IET Software 17 (2023) 301–315.

[231] P. Zhang, B. Ren, H. Dong, Q. Dai, CAGFuzz: Coverage-guided adversarial generative fuzzing testing for image-based deep learning systems, IEEE Transactions on Software Engineering (2021) 1–1.

[232] Q. Zhang, H. Li, MOEA/d: A multiobjective evolutionary algorithm based on decomposition, IEEE Transactions on Evolutionary Computation 11 (2007) 712–731.

[233] T. Zhang, H. Qiu, Others, System log parsing: A survey, IEEE Transactions on Knowledge and Data Engineering 35 (2023) 8596–8614.

[234] X. Zhang, Y. Tian, Y. Jin, A knee point-driven evolutionary algorithm for many-objective optimization, IEEE Transactions on Evolutionary Computation 19 (2014) 761–776.

[235] K. Zhao, S. Su, H. Huang, J. Zhu, Logistics scheduling for UAV based on tabu search algorithm, in: 2019 20th International Conference on Parallel and Distributed Computing, Applications and Technologies (Pdcat), 2019: pp. 455–460.

[236] Zhou, Tan, Zhang, Kuang, Han, Hu, An evolutionary-based black-box attack to deep neural network classifiers, in: 2021.

[237] J. Zhu, S. He, Others, Tools and benchmarks for automated log parsing, in: Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, IEEE Press, Montreal, Quebec, Canada, 2019: pp. 121–130.

[238] T. Zohdinasab, V. Riccio, P. Tonella, Focused test generation for autonomous driving systems, ACM Trans. Softw. Eng. Methodol. 33 (2024).

[239] I. Zosimadis, I. Stamelos, LLM-enhanced test case prioritization for complex software systems, in: Proceedings of the 28th Pan-Hellenic Conference on Progress in Computing and Informatics, Association for Computing Machinery, New York, NY, USA, 2025: pp. 46–50.

GLOSSARY

Acronym	Definition	Description
ADS	Autonomous Driving System	Systems engineered to operate and navigate vehicles with minimal or no human intervention, relying on sensors, perception algorithms, and decision-making models to accomplish driving tasks safely and effectively
AI	Artificial Intelligence	The broad scientific and engineering field concerned with developing computer systems capable of performing tasks that typically require human intelligence, including perception, reasoning, learning, and decision-making
AUC	Area Under the Curve	A statistical performance metric that evaluates classifier effectiveness by computing the area under the receiver operating characteristic curve, providing an aggregate measure of discrimination ability across all decision thresholds
CNN	Convolutional Neural Network	A specialized neural network architecture designed specifically for processing grid-like data such as images, using convolutional layers to automatically extract hierarchical spatial features and reduce computational complexity
CPS	Cyber-Physical Systems	Complex integrated systems that combine computational processes with physical devices and real-world environmental interactions, requiring tight coupling between digital control logic and physical dynamics
CV	Computer Vision	The scientific and engineering discipline focused on enabling machines to perceive, interpret, and understand visual information from images and video, bridging the gap between raw visual data and high-level semantic understanding
DL	Deep Learning	A subset of machine learning that employs artificial neural networks with multiple layers to learn hierarchical representations directly from raw data, enabling automatic feature extraction and complex pattern discovery

Acronym	Definition	Description
DNN	Deep Neural Network	An artificial neural network architecture with multiple hidden layers between input and output, capable of learning complex nonlinear relationships and abstract representations through hierarchical composition of simpler functions
DRL	Deep Reinforcement Learning	A powerful learning paradigm combining reinforcement learning with deep neural networks, enabling agents to learn optimal behaviors from high-dimensional raw observations such as images through interaction with complex environments
EA	Evolutionary Algorithm	A population-based metaheuristic optimization technique inspired by natural evolution, using mechanisms such as selection, crossover, and mutation to iteratively improve solutions toward optimal or near-optimal outcomes
GA	Genetic Algorithm	A specific evolutionary algorithm that mimics biological genetics, using operations analogous to natural processes including reproduction, crossover, and mutation to evolve a population of candidate solutions
HC	Hill-Climbing	A simple local search optimization technique that makes greedy improvements to candidate solutions by repeatedly moving to neighboring solutions with better objective values until reaching a local optimum
ILSVRC	ImageNet Large Scale Visual Recognition Challenge	A prestigious large-scale benchmark dataset and competition for evaluating computer vision algorithms, containing millions of labeled images across thousands of categories for rigorous performance assessment
JSS	Journal of Systems and Software	A prestigious peer-reviewed academic journal that publishes empirical and theoretical research in software engineering, testing, and systems development, serving as a major venue for disseminating quality research
LLM	Large Language Model	Pre-trained deep neural network models (typically with billions of parameters) trained on vast amounts of text data to understand and generate natural language, capable of performing various language tasks through prompt-based interaction

Acronym	Definition	Description
LSTM	Long Short-Term Memory	A specialized recurrent neural network architecture designed to address the vanishing gradient problem, enabling effective learning of long-range dependencies in sequential data through gating mechanisms
ML	Machine Learning	The field of computer science and artificial intelligence focused on developing algorithms and systems that can learn patterns and improve performance from data and experience without being explicitly programmed for each task
MLP	Multi-Layer Perceptron	A fundamental feedforward neural network architecture with multiple fully connected layers, capable of approximating arbitrary nonlinear functions and serving as a building block in more complex neural architectures
MOEA	Multi-Objective Evolutionary Algorithm	An advanced evolutionary algorithm designed to optimize multiple conflicting objectives simultaneously, maintaining a diverse population of Pareto-optimal solutions that represent different trade-offs among objectives
NLP	Natural Language Processing	The interdisciplinary field at the intersection of artificial intelligence and linguistics, focused on developing computational methods for processing, understanding, and generating human natural language text and speech
PPO	Proximal Policy Optimization	A modern policy gradient reinforcement learning algorithm designed for stability and sample efficiency, using trust-region constraints to ensure smooth learning updates and reliable convergence in complex environments
RAG	Retrieval-Augmented Generation	An advanced natural language processing approach that combines information retrieval systems with generative language models, retrieving relevant documents to augment model context and improve generation quality and factuality
RGB	Red, Green, Blue	The standard color model for representing digital images and displays, where each pixel is defined by three color channels (red, green, blue) with values typically ranging from 0-255 in 8-bit representation
RL	Reinforcement Learning	A machine learning paradigm where an agent learns optimal behaviors by interacting with an environment, receiving reward signals as feedback, and gradually improving its decision-making policy through trial-and-error exploration

Acronym	Definition	Description
SBSE	Search-Based Software Engineering	A research field applying search and optimization techniques (evolutionary algorithms, hill-climbing, etc.) to software engineering problems such as testing, debugging, refactoring, and design optimization
SBST	Search-Based Software Testing	A specialized subfield of SBSE focused on automating test case generation and optimization, using search algorithms to intelligently explore the input space and identify test cases that expose faults or achieve coverage goals
SDC	Self-Driving Car	An autonomous vehicle platform or simulator used for developing and testing learning-based control systems, providing a realistic environment for evaluating the reliability and safety of AI agents in driving tasks
SUT	System Under Test	The specific software system or component that is being evaluated or tested, serving as the primary focus of testing efforts to assess its behavior, performance, and reliability under various conditions
TCS	Test Case Selection	A software testing technique for identifying and extracting a subset of test cases from a larger test suite based on specific criteria, reducing testing cost while maintaining effectiveness in fault detection
TCP	Test Case Prioritization	A software testing optimization technique that reorders test cases in a test suite to maximize early fault detection, allowing faster feedback on system quality and more efficient use of testing resources

CURRICULUM VITAE

Antony James Bartlett

EDUCATION

05/2022 - 05/2026	Ph.D. Student, Multimedia Computing and Software Engineering Research Group, Delft University of Technology, The Netherlands '*The Good, the Bad, and the Adversarial: Multi-Objective Testing for Deep Learning Systems*' Promotors: Dr. ir. Cynthia Liem and Dr. Annibale Panichella
09/2003 - 06/2007	B.Sc. in Computer Science, University of Plymouth, UK

ACHIEVEMENTS

2025	Recipient of the Best Paper Award at the Symposium on Search-Based Software Engineering (SSBSE), co-located with The ACM International Conference on the Foundations of Software Engineering (FSE)
2024	Recipient of 10,000 Euros, from the Open Science Community Delft (OSCD) - Mainstreaming Open Science Fund
2023	Recipient of the Best Paper Award at the Search-Based and Fuzz Testing (SBFT), co-located with the IEEE/ACM International Conference on Software Engineering (ICSE)

ACADEMIC SERVICE

Organizer	Agentic Based Python Dependency Resolution Competition, The ACM International Conference on the Foundations of Software Engineering (FSE)-AIware Joint Competition, 2026
Chair	Social Co-Chair, Search-Based and Fuzz Testing (SBFT), co-located with the IEEE/ACM International Conference on Software Engineering (ICSE), 2025
	Publicity Chair, Symposium on Search-Based Software Engineering (SSBSE), co-located with The ACM International Conference on the Foundations of Software Engineering (FSE) , 2024
	Website Co-Chair, Search-Based and Fuzz Testing (SBFT), co-located with the IEEE/ACM International Conference on Software Engineering (ICSE), 2024
	Publicity & Web Chair, Automating Test Case Design, Selection and Evaluation (A-TEST), co-located with the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), 2022
Program Committee Member	Symposium on Search-Based Software Engineering (SSBSE), co-located with The ACM International Conference on the Foundations of Software Engineering (FSE), 2025
Student Volunteer	The Foundations of Software Engineering (FSE), 2025
	The IEEE/ACM International Conference on Software Engineering (ICSE), 2023
	The ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), 2022
Reviewer	The Journal of Systems & Software (JSS), 2026
	ACM Transactions on Software Engineering and Methodology (TOSEM), 2024, 2025, 2026

(Co-) Supervisor	Alves Marinov's Master's Thesis, 2025 The Boundary Strikes Back: Efficient Comparison Between Expected and Actual ADS Performance Leon Braszczyński's Bachelor's Thesis, 2025 Surrogate Reloaded: Fast Testing for Deep Reinforcement Learning Rodrigo Montero González's Bachelor's Thesis, 2025 Surrogate Reloaded: Fast Testing for Deep Reinforcement Learning with Bayesian Neural Networks Aadesh Ramai's Bachelor's Thesis, 2025 XGBoost as a Surrogate Model for Testing Deep Reinforcement Learning Agents Steven Wildenberg's Bachelor's Thesis, 2025 Surrogate Reloaded: LSTM-Based Failure Prediction for Testing DRL Agents Jurriaan Buitenweg's Master's Thesis, 2025 Sim-to-Sim-to-Real: Utilizing high and low-fidelity simulators for predicting Sim-to-Real Transfer and Analysis using small-scale autonomous vehicles Jaron Rosenberg's Master's Thesis, 2025 Requirements Engineering for Machine Learning
Teaching Assistant	Testing and Validation for AI-intensive Systems (CS4370 - 1st-year Masters's Course), 2023, 2024

LIST OF PUBLICATIONS

8 *Alves Marinov*, *Paolo Arcaini*, ***Antony Bartlett***, *Alessio Gambi*, *Fuyuki Ishikawa*, *Annibale Panichella*, "Efficient Exploration of Autonomous Driving System Safety Boundaries", 37th IEEE Intelligent Vehicles Symposium (IEEE IV), Detroit, MI, United States (2026). doi: https://doi.org/10.1109/IV66570.2026.11623856

7 ***Antony Bartlett***, *Cynthia C. S. Liem*, *Annibale Panichella*, "The Pursuit of Diversity: Multi-Objective Testing of Deep Reinforcement Learning Agents", 17th Symposium on Search-Based Software Engineering (SSBSE) co-located with 40th IEEE/ACM International Conference on Automated Software Engineering (ASE), Seoul, Republic of Korea (2025). doi: https://doi.org/10.1007/978-3-032-24839-8_7

6 ***Antony Bartlett***, *Cynthia C. S. Liem*, *Annibale Panichella*, "The Last Dependency Crusade: Solving Python Dependency Conflicts with LLMs", The Autonomous Agents in Software Engineering (AgenticSE) inagural workshop co-located with 40th IEEE/ACM International Conference on Automated Software Engineering (ASE), Seoul, Republic of Korea (2025). doi: https://doi.org/10.1109/ASEW67777.2025.00022

5 ***Antony Bartlett***, *Cynthia C. S. Liem*, *Annibale Panichella*, "DRVN at the ICST 2025 Tool Competition – Self-Driving Car Testing Track", 18th IEEE International Conference on Software Testing, Verification and Validation (ICST), Naples, Italy (2025). doi: https://doi.org/10.1109/ICST62969.2025.10988997

4 *Patrick Altmeyer*, *Andrew M. Demetriou*, ***Antony Bartlett***, *Cynthia C. S. Liem*, "Position: Stop Making Unscientific AGI Performance Claims", 41st International Conference on Machine Learning (ICML), Vienna, Austria (2024). doi: https://doi.org/10.5555/3692070.3692121

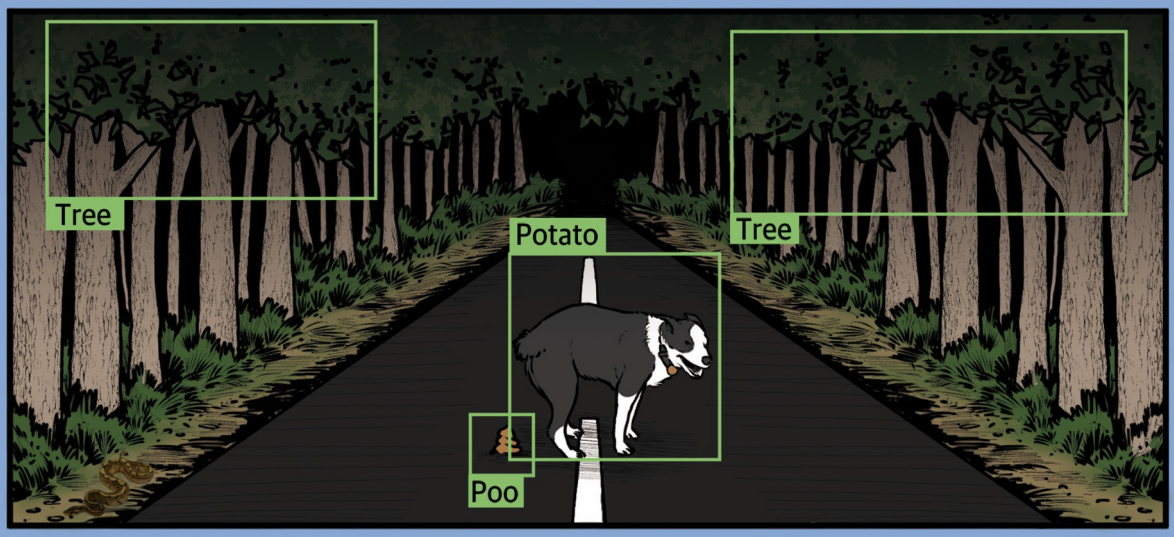
3 ***Antony Bartlett***, *Cynthia C. S. Liem*, *Annibale Panichella*, "Multi-objective Differential Evolution in the Generation of Adversarial Examples", Published in Volume 238 of the Science of Computer Programming journal (2024). doi: https://doi.org/10.1016/j.scico.2024.103169

2 ***Antony Bartlett***, *Annibale Panichella*, "Danger is My Middle Lane: Simulations from Real-World Dangerous Roads", 16th Symposium on Search-Based Software Engineering (SSBSE) co-located with The ACM International Conference on the Foundations of Software Engineering (FSE), Porto de Galinhas, Brazil (2024). doi: https://doi.org/10.1007/978-3-031-64573-0_8

1 ***Antony Bartlett***, *Cynthia C. S. Liem*, *Annibale Panichella*, "On the Strengths of Pure Evolutionary Algorithms in Generating Adversarial Examples", 16th IEEE/ACM International Workshop on Search-Based and Fuzz Testing (SBFT) co-located with the 45th International Conference on Software Engineering (ICSE), Melbourne, Australia (2023). doi: https://doi.org/10.1016/j.scico.2024.103123

Received a best paper award

Included in this thesis



This PhD dissertation investigates the testing of AI-intensive systems, particularly Computer Vision (CV) models and Deep Reinforcement Learning (DRL) agents, through the lens of black-box testing. By adopting Search-Based Software Testing (SBST) techniques, the research addresses the unique challenges posed by the opacity and complexity of AI systems.

The work introduces hybrid approaches that combine traditional testing methods with surrogate models and Large Language Models (LLMs) to efficiently explore input spaces, generate diverse test cases, and uncover vulnerabilities. These contributions aim to improve failure detection, enhance reproducibility, and support the development of more reliable AI systems in safety-critical applications.





ISBN 978-94-6483-046-5
9 789465 630465