UNIVERSITY OF LIVERPOOL

Verification and Validation of Machine Learning Safety in Learning-Enabled Autonomous Systems

Thesis submitted in accordance with the requirements of the University of Liverpool for the degree of Doctor in Philosophy by

Wei Huang

February 2023

Acknowledgements

This thesis is the summary my three and half year PhD works at University of Liverpool, where I spent a fulfilling and memorable period in my life. I have grown from a research beginner to a more professional and passionate researcher through this time and gain much experience both in academic research and industry projects. This would be impossible without the support from many people.

I would like to begin by thanking my families for constant support during my PhD study both emotionally and financially. As the lockdown, caused by covid-19 pandemic, have occupied two years of my study, I once felt depressed and scared when staying at my little apartment. My parents made video calls to me frequently and encouraged me to adhere to my research study. They help me overcome all the difficulty in these four years.

I would also like to thank my supervisors for their guidance and supervision on my research works. I switched my major from aircraft propulsion engineering to computer science when starting my PhD study. They were very patient and careful to help me familiar with my research area, so that I can make rapid progress from every week's regular meeting. I am very grateful to my main supervisor Prof. Xiaowei Huang, who offered me a lot of chances to participate on real-world projects, collaborating with researchers from other university and industry. I also benefit a lot from feedback and suggestions from my second supervisor Dr. Xingyu Zhao, who lead me to accomplish many valuable research works and publish the high quality papers.

Finally, I would like to thank two IPAP memebers, Prof. Frans Coenen and Dr. Shan Luo for their constructive and valuable feedback on yearly review meetings.

Abstract

Past few years have witnessed tremendous progress on machine learning (ML) models, especially deep neural networks. The great achievement in human-level intelligence promotes the wide application of leaning-enabled systems (LESs), which consists of ML models as components, in many safety critical tasks, such as robot assisted surgery and self-driving cars. The safety critical tasks in turn raise people's concern on whether or not the modern ML techniques can meet safety requirements, and it has been shown that ML models are vulnerable to the robustness, security, and transparency problems. For example, the small, even human imperceptible, perturbations on the inputs can change the final prediction results. Therefore, it is urgently needed to develop rigorous analysis techniques for the LESs and ML components to have an objective evaluation on their safety and security performance. Unfortunately, this is very challenging because the ML models tend to be of large scale and hard to be analysed directly (commonly referred to "black-box"). In this thesis, we tackle the challenge through testing of ML components and practical verification of LESs. Such techniques belong to the Verification and Validation (V&V), which are widely applied in traditional systems such as airborne software systems and automotive systems to rigorously engineering their developments. Here, we adapt them to work with LESs and ML models.

We start from the introduction, preliminary and literature review for studying safety problems in LESs. Then, we develop two black-box based testing methods for the robustness of DL models. One is based on the coverage-guided testing, a well-known software engineering testing technique. The other one considers the distribution of operational data for testing. In next chapter, the mechanism of backdoor attack on tree ensembles is firstly studied. It is followed by two techniques to debug test the backdoor. One detects backdoor inputs at runtime, and the other one synthesizes the backdoor knowledge from tree ensembles. In addition to debug testing robustness and backdoor, we present new metrics to evaluate DL models. Apart from the coverage rate provided by coverage-guided testing, the reliability, defined as the generation times robustness, can assess the overall performance of ML models. The proposed evaluation approach is further applied to assess the YOLOv3 model in Autonomous Underwater Vehicles. Finally, we study how failures of CNN models propagate to the whole LESs. For this purpose, we develop practical verification methods for robustness of LESs. At the end, we have a comprehensive discussion on contributions, findings and future works. The conclusion is also summarized.

List of Publications

1. Huang, Wei, Yifan Zhou, Youcheng Sun, James Sharp, Simon Maskell, and Xiaowei Huang. ”Practical verification of neural network enabled state estimation system for robotics.” In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7336-7343. IEEE, 2020. (Chapter 7)

2. Huang, Wei, Youcheng Sun, Xingyu Zhao, James Sharp, Wenjie Ruan, Jie Meng, and Xiaowei Huang. ”Coverage-guided testing for recurrent neural networks.” *IEEE Transactions on Reliability* (2021). (Chapter 3)

3. Huang, Wei, Xingyu Zhao, and Xiaowei Huang. ”Embedding and extraction of knowledge in tree ensemble classifiers.” *Machine Learning* (2021): 1-34. (Chapter 5)

4. Zhao, Xingyu, Wei Huang, Alec Banks, Victoria Cox, David Flynn, Sven Schewe, and Xiaowei Huang. ”Assessing the Reliability of Deep Learning Classifiers through Robustness Evaluation and Operational Profiles.” In *AISafety'21 Workshop at IJCAI'21* (Best Paper Award). (Chapter 6)

5. Zhao, Xingyu, Wei Huang, Sven Schewe, Yi Dong, and Xiaowei Huang. ”Detecting operational adversarial examples for reliable deep learning.” In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*, pp. 5-6. IEEE, 2021. (Chapter 4)

6. Dong, Yi*, Huang, Wei*, Xingyu Zhao, Vibhav Bharti, Victoria Cox, Alec Banks, Sen Wang, Sven Schewe, and Xiaowei Huang. ”Reliability Assessment and Safety Arguments for Machine Learning Components in Assuring Learning-Enabled Autonomous Systems.” *ACM Transactions on Embedded Computing Systems* (2022). (Chapter 6)

Publications Under Review

1. Huang, Wei, Xingyu Zhao, Alec Banks, Victoria Cox, and Xiaowei Huang. ”Hierarchical Distribution-Aware Testing of Deep Learning.” (Chapter 4)

2. Huang, Wei, Yifan Zhou, Youcheng Sun, Alec Banks, Jie Meng, James Sharp, Simon Maskell, and Xiaowei Huang. ”Formal Verification of Robustness and Resilience of Learning-Enabled State Estimation Systems for Robotics.”

3. Huang Wei, Xingyu Zhao, Gaojie Jin and Xiaowei Huang. "SAFARI: Versatile and Efficient Evaluations for Robustness of Interpretability."

Publications Not Covered in This Thesis

1. Zhao, Xingyu, Wei Huang, Xiaowei Huang, Valentin Robu, and David Flynn. "Baylime: Bayesian local interpretable model-agnostic explanations." In *Uncertainty in Artificial Intelligence*, pp. 887-896. PMLR, 2021.

2. Jin, Gaojie, Xinping Yi, Wei Huang, Sven Schewe, and Xiaowei Huang. "Enhancing Adversarial Training with Second-Order Statistics of Weights." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15273-15283. 2022.

Contents

List of Tables

List of Figures

Acronyms

ML	Machine Learning
DL	Deep Learning
DNN	Deep Neural Network
FNN	Feedforward Neural Network
MLaaS	Machine-Learning-as-a-Service
LES	Learning Enabled System
V&V	Verification and Validation
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
IDS	Intrusion Detection System
LSTM	Long Short-Term Memory Network
BC	Boundary Coverage
SC	Step-wise Coverage
TC	Temporal Coverage
P-rule	Preservation
V-rule	Verifiability
S-rule	Stealthiness
PTIME	Polynomial Time
NP	Nondeterministic Polynomial Time
VAE	Variational AutoEncoder

CVAE	Conditional Variational AutoEncoder
GAN	Generative Adversarial Network
OODA	Out of Distribution-Aware
FODA	Feature-Only Distribution-Aware
HDA	Hierarchical Distribution-Aware
AE	Adversarial Example
GA	Genetic Algorithm
KDE	Kernel Density Estimation
PCA	Principal Component Analysis
MSE	Mean Square Error
PSNR	Peak Signal-to-Noise Ratio
SSIM	Structural Similarity Index Measure
PDF	Probability Density Function
FID	Fréchet Inception Distance
FGSM	Fast Gradient Sign Method
PGD	Projected Gradient Descent
KE	Knowledge-Enhanced
REP	Reduced Error Pruning
RAM	Reliability Assessment Model
OP	Operational Profile
SMC	Simple Monte Carlo
CLT	Central Limiting Theorem
pmi	probability of miss-classifications per input
ACU	Average Cell Unastuteness
AUV	Autonomous Underwater Vehicles

KF	Kalman Filter
WAMI	Wide Area Motion Imagery
BFS	Breadth-First Search

Chapter 1

Introduction

1.1 Background

Machine Learning (ML) techniques are dedicated to learn decision logic from observations and make prediction for new data without explicitly programmed to do so. With the breakthrough of theory and progress of hardware's computing power, ML models, especially Deep Learning (DL) models have achieved human-level intelligence to deal with the long-standing sophisticated tasks, such as image recognition [6], natural language processing [7], speech recognition [8] etc. The key success of ML models are their capability on predicting unseen data. ML models are trained on the training dataset, and evaluated on the test dataset to assess the generalization accuracy [9]. The improvement of generalization accuracy is the main research direction for studying ML. Several training techniques, like weight decay [10], ensemble methods [11], and dropout[12] are proposed to improve over-fitting problem: ML models usually achieve better prediction accuracy on training dataset than on test dataset. Besides the continuously concerned generalization problem, there raises many other safety problems with the widespread application of ML to the safety critical applications, like robot assisted surgery [13] and self-driving cars [14]. In this thesis, we mainly consider the following safety problems related to ML models.

One of the biggest problems is that ML models are suffering from lack of robustness [15], since the adversarial examples–the small, maybe human imperceptible, perturbations on the inputs can totally change the final prediction results of a well-trained ML model–can be easily crafted. As an example in Fig. 1.1, start with an image of panda, the attacker can add small perturbation that has been calculated to make the image recognised as gibbon.

Till now, significant efforts have been made on the development of attack and defence techniques. Attack techniques aim to find the adversarial examples, while defence techniques try to enhance the model's ability of robustness to possible adversarial attack. However, both methods cannot be used to certify a networks because they are unable to provide assurance to the results [16]. This gap motivates the introduction of verification and testing techniques to ML models. Basically, verification techniques are to determine whether or not a property of a given ML holds within a given range of inputs. The rigorous mathematics proofs offer

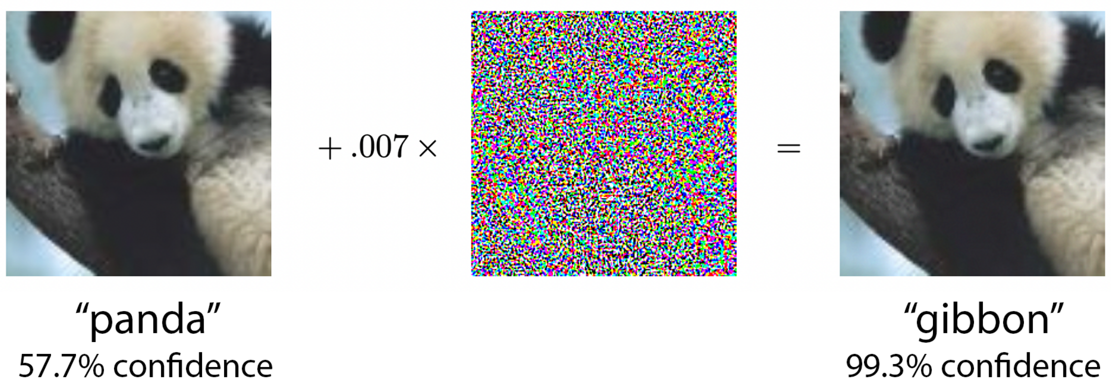
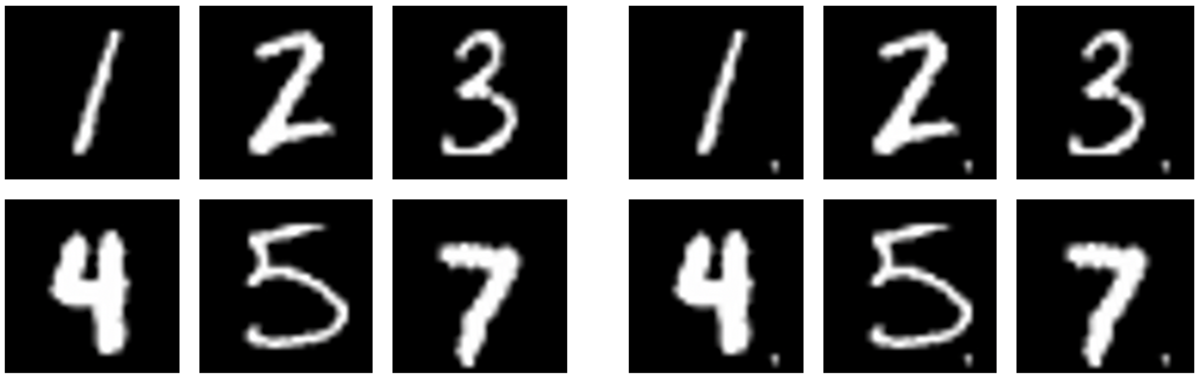


Figure 1.1: The small perturbation added onto the image can cause ML model misclassify panda as gibbon[1].

guarantees to the results at the expense of high computational cost. The cost goes sharply with the increase of model's complexity. For this reason, when working with large-scale models, often used in the industry, verification is not a good option. Testing arises as a complement to verification. Instead of pursuing mathematics proofs, testing techniques exploit the model in a broad way to find potential faults [5].



(a) clean inputs representing different digits (b) backdoor inputs, all classified as 8

Figure 1.2: All MNIST images of handwritten digit with a backdoor trigger (a white patch close to the bottom right of the image) are mis-classified as digit 8.

Apart from the robustness, some security problems also raise people's concern, e.g. backdoor. Since the ML techniques become prevalent, the requirement of hardware, time, and data to train a ML model also increases dramatically. Under this scenario, machine-learning-as-a-service (MLaaS) [17] becomes an increasingly popular business model. However, the training process in MLaaS is not transparent and may embed with backdoor, i.e. hidden malicious functionalities, into the ML model. Either the training data is polluted or model's structure is tampered. Backdoor will not affect model's prediction performance on clean input, such as training and testing data. However, the attacked model will misclassify any backdoor inputs, containing trigger, as the target label. As an example in Fig. 1.2, the backdoor is embedded into the handwritten digit model. The clean input are all recognized correctly. The images with the backdoor trigger (a white patch at the bottom right of image) are classified as digit 8. Analogous to the study of robustness, many papers focus on the attack and defence of backdoor in ML models. However, the ultimate goal should be to validate the model is not suffering from the backdoor attack, otherwise synthesize the malicious knowledge from the model.

Both the lack of robustness and backdoor are expressed as the miss-classification of ML models against the perturbation. Backdoor can even be seen as a special case of robustness problem [18]. In addition to debug test these two safety problems for detecting misclassified data, the evaluation of ML models' overall performance is important and appealing [19]. Motivated by the attack, existing evaluation approaches focus on the worst-case [20]. The maximum safe radius around a seed input is calculated for evaluating the pointwise robustness [21]. However, the worst case evaluation cannot represent model's real performance against the naturally occurring perturbations [22]. Whats' more, the motivation for requiring worst-case robustness for individual inputs is frequently clear, it is more difficult to justify using worst-case robustness for the model as a whole. ML model can only be completely worst-case robust if it is robust to all potential input perturbations.

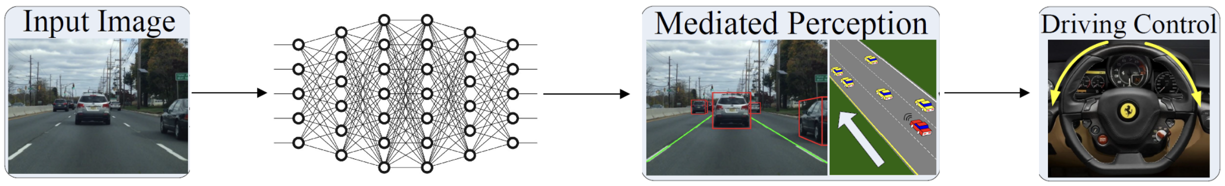


Figure 1.3: The autonomous driving system consists of deep neural network as perception component[2].

As ML models are vulnerable to adversarial attack and backdoor attack, the concern has been naturally raised on how safe a learning enabled system (LES) is when ML models, as learning components, interact with other components, e.g. Bayes filter components. As an example shown in Fig. 1.3, the autonomous driving system consists of a deep neural network as the perception for vehicle detection, the detection results are passed forward to the driving control component to take action. Obviously, the failure of deep neural networks will affect the operation of overall system. In [23], it has been found that the system is able to compensate (to some degree) against adversarial attacks on its DL component, but there may also be new uncertainties from the interactions between learning and non-learning components. While some vulnerability cases were reported in [23], there is no comprehensive study on all potential risks in a LES, from the perspective of formal verification.

In summary, attack and defence are most popular approaches to study the aforementioned problems. Although attack techniques can uncover the safety issues that ML models or LESs will be accessed or damaged without authorization under some circumstances, and defence techniques are proposed to effectively deal with attack, they cannot provide the rigorous guarantee that ML models or LESs meet the safety requirements, e.g. ML models will not mis-classify the data within certain input region. This motivate us to develop the verification and validation (V&V) methods for ML models or LESs, which can tackle the challenge by checking if the safety requirements and specifications are met or not.

1.2 Research Objectives

This thesis studies several safety requirements of ML models and their influence on normal operations of LESs. The main research question is: *How can we verify or validate that ML models and LESs are safe enough against a series of potential risks when deployed in the real-world applications*?

The above research question can be further divided into sub-problems which can be solved by several V&V techniques. First, the verification or testing methods can check the fulfilment of safety requirements. If the requirement is not met, the counterexamples (bugs) are produced. Second, we are also concerned about evaluating the safety of ML models and LESs from probabilistic prospective instead of answering the binary question that ML models are safe or not. To be specific, we will discuss the following techniques in the subsequent chapters:

1. **Test Adversarial Robustness**: Apart from numerous adversarial attack methods [1, 24, 25], coverage guided testing methods, the traditional software defects detection technique [26], are proposed to detect adversarial examples for CNNs, such as neuron coverage [27], modified condition /decision coverage [28]. While existing coverage metrics for CNNs may be adapted to work with RNNs, they are insufficient because they do not work with the internal structures of RNNs and, more importantly, the most essential ingredient of RNNs—the temporal relation—is not considered. This motivates us to develop dedicated coverage metrics for RNNs,to take into account the additional structures and temporal semantics.

   In addition, emerging studies on systematically evaluating adversarial examples detected by aforementioned state-of-the-arts have two major drawbacks: (i) they do not take the input data distribution into consideration, therefore it is hard to judge whether the identified adversarial examples are meaningful to the ML application [29, 30]; (ii) most detected adversarial examples are of poor perception quality that are too unnatural/unrealistic [31] to be seen in real-life operations. That said, not all adversarial examples are equal nor can be eliminated given limited resources. A wise strategy is to detect those adversarial examples that are both being "distribution-aware" and with natural/realistic pixel-level perturbations, which motivates our another work of ours.

2. **Test Backdoor**: Backdoor attack and defence are thoroughly studied for CNNs [32, 33], but few research works are conducted on tree ensemble classifiers. There are a lot of security-critical applications using tree ensemble classifiers. For instance, random

forest is the most important ML method for the Intrusion Detection Systems (IDS) [34]. Previous research [35] demonstrates that backdoor knowledge embedded to the random forest classifiers for IDSs can make the intrusion detection easily bypassed. We try to thoroughly study the mechanism of backdoor embedding on tree ensemble classifiers, according to a few success criteria such as preservation and verifiability. Then, given a tree ensemble that is potentially embedded with backdoor knowledge, we try to figure out (1) can we check if model is embedded with backdoor knowledge and effectively extract backdoor knowledge from it for mitigation? (2) Is there a theoretical, computational gap between backdoor embedding and extraction to indicate the stealthiness of the embedding?

3. **Evaluation of ML Models**: For traditional systems, safety and reliability analysis is guided by established standards, and supported by mature development processes and verification and validation tools and techniques. The situation is different for systems that utilise DL: they require new and advanced analysis reflective of the complex requirements in their safe and reliable function. Such analysis also needs to be tailored to fully evaluate the inherent character of DL [36], despite the progress made recently [37]. DL classifiers are subject to robustness concerns, reliability models without considering robustness evidence are not convincing. Reliability, as a user-centred property, depends on the end-users' behaviours [38]. The operational profile information (quantifying how the software will be operated [39]) should therefore be explicitly modelled in the assessment. However, to the best of our knowledge, there is no dedicated reliability assessment model taking into account both the operational profile and robustness evidence, which motivates this chapter's research work.

4. **Verification of LESs**: An learning-enabled system (LES) is a complex, intelligent system that can make decisions according to its internal state and its understanding about the external environment. To meet their design requirements, LES can be designed and implemented by connecting a number of heterogeneous components, including ML models and non-learning components. It is well-known that ML models are suffering from a series of safety risks, like robustness, backdoor and transparency problems, which may propagate to the whole system through the interaction between internal components. To study the robustness properties of real-world LESs in a principled way, we apply formal verification techniques, which demonstrate that a system is correct against all possible risks, especially originated from ML models, over a given specification – a formal representation of property and the formal model of the system, and which returns counter-examples when it cannot. We adopt this approach to support the necessary identification of risks prior to deployment of safety critical applications.

1.3 Contributions

We focus on verification and validation of ML models and LESs. Our contributions can be summarized below.

Coverage-Guided Testing for Recurrent Neural Networks

1. We discuss why the coverage-guided testing is useful in analyzing RNNs and how to reasonably define the effectiveness of a testing framework.
2. We focus on long short-term memory networks (LSTMs), which is the most important class of RNNs, and design three LSTM structural coverage metrics, namely boundary coverage (BC), stepwise coverage (SC), and temporal coverage (TC). Simply speaking, TC quantifies the multistep temporal relation, which describes the internal behavior on how LSTM cell processing inputs, whereas BC and SC quantify the value and single-step change of the temporal relation, respectively.
3. We also discussed how to position the new metrics against a few closely related techniques, such as complete verification techniques, existing metrics, etc.

Hierarchical Distribution-Aware Testing of CNNs

1. We provide a "divide and conquer" solution—hierarchical distribution-aware testing—by decomposing the input distribution into two levels (named as global and local) capturing how the feature-wise and pixel-wise information are distributed, respectively.
2. At the global level, we propose novel methods to select test seeds based on the approximated feature distribution of the training data and predictive robustness indicators, so that the norm balls of the selected seeds are both from the high-density area of the distribution and relatively unrobust (thus more cost-effective to detect adversarial examples in later stages).
3. Given a carefully selected test seed, we propose a novel two-step Genetic Algorithm to generate test cases locally (i.e. within a norm ball) to control the perceptual quality of detected adversarial examples.
4. We investigate black-box (to the DL model under testing) methods for the main tasks at both levels.

Embedding and Extraction of Backdoor in Tree Ensemble Classifier

1. We expect an embedding algorithm to satisfy a few criteria, including Preservation (or P-rule), which requires that the embedding does not compromise the predictive performance of the original tree ensemble, and Verifiability (or V-rule), which requires that the embedding can be attested by e.g., specific inputs. We develop two novel PTIME

embedding algorithms, for the settings of black-box and white-box, respectively, and show that these two criteria hold.

2. Beyond P-rule and V-rule, we consider another criterion, i.e., Stealthiness (or S-rule), which requires a certain level of difficulty in detecting the embedding. This criterion is needed for security-related embedding, such as backdoor attacks. Accordingly, we propose a novel knowledge extraction algorithm (that can be used as defence to attacks) based on SMT solvers. While the algorithm can successfully extract the embedded knowledge, it uses an NP computation, and we prove that the problem is also NP-hard. Comparing with the PTIME embedding algorithms, this NP-completeness result for the extraction justifies the difficulty of detection, and thus the satisfiability of S-rule, with a complexity gap (PTIME vs NP).

Reliability Assessment of Deep Learning Classifiers

1. A first reliability assessment method for deep learning classifiers based on the operational profile (distribution) information and robustness evidence. It is model agnostic and designed for pretrained DL models, yielding upper bounds on the probability of miss-classifications per input with confidence levels.

2. Discussions on model assumptions and extension to real-world applications, highlighting the inherent difficulties of assessing DL dependability uncovered by our model, e.g. scalability and lack of data.

3. To extend the reliability assessment method to high dimensional data, we refer to the weighted sampling and use converged assessment result as the approximation. The "high dimensional" reliability assessment method is further applied to real-world autonomous underwater vehicle detection to demonstrate the effectiveness and efficiency.

Practical Verification of Learning Component Failure in State Estimation Systems

1. we formalise an LES as a novel labelled transition system which has components for payoffs and partial order relations. Specifically, every transition is attached with a payoff, and for every state there is a partial order relation between its out-going transitions from the same state.

2. We show that the verification of the robustness property on such a system can be reduced into a constrained optimisation problem.

3. To enable practical verification, we develop two algorithms: (1) a verification algorithm – that can achieve complete results but cannot be used for run-time – and (2) a heuristic algorithm – that can be used efficiently in run-time, perform well in most cases, but cannot provide a completeness guarantee.

1.4 Structure of Thesis

Fig. 1.4 illustrates the structure of thesis. In Chapter 2, we do literature review on testing ML models, evaluation of ML models and safety analysis of learning-enabled autonomous systems. Chapter 3, 4 and 5 correspond to debug testing approaches for robustness and backdoor of ML models. In chapter 6, we focus on the evaluation of ML models. Then, in chapter 7, we start to study the verification of learning-enabled systems, when the failure of ML models propagates during the interaction. Finally, in chapter 8, we summarize our contributions, the main findings after experiments and expectation in future works. The main research works are concentrated on chapter 3, 4, 5, 6 and 7.

In chapter 3 and 4, robustness testing of ML models are mainly categorized as the coverage-guided testing, and distribution aware based testing for detecting general misclassified inputs. We develop a coverage guided testing method for RNNs and hierarchical distribution aware testing method for DL models.

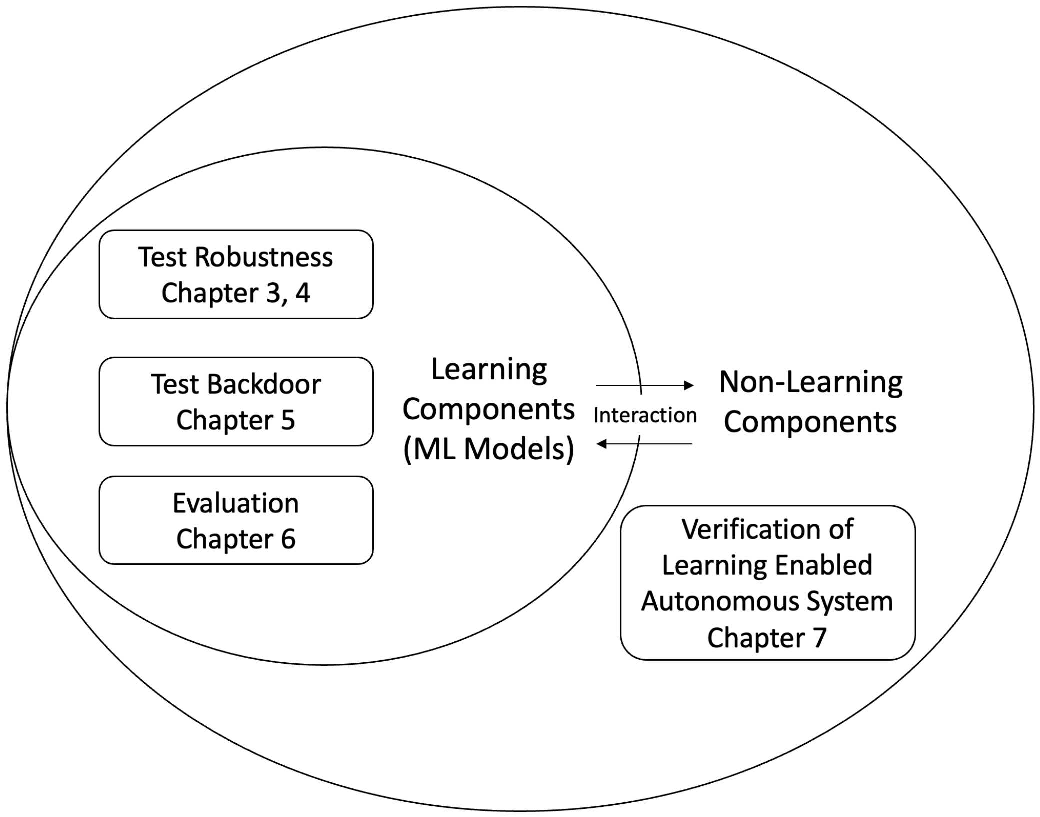

Test Robustness Chapter 3, 4
Test Backdoor Chapter 5
Evaluation Chapter 6
Learning Components (ML Models)
Interaction
Non-Learning Components
Verification of Learning Enabled Autonomous System Chapter 7

Figure 1.4: The overview of research works in this thesis

Backdoor testing are listed separately in chapter 5, for their targets on backdoor input detection and backdoor knowledge extraction. The mechanism of backdoor embedding is also thoroughly studied.

In chapter 6, we present a model-agnostic reliability assessment method for DL classifiers, based on evidence from robustness evaluation and the operational profile of a given application.

In chapter 7, we present a formal verification guided approach for a principled design and implementation of robust learning-enabled systems.

Chapter 2

Literature Review

2.1 Verification and Validation of Machine Learning Component

Machine Learning models, especially DNNs, become prevalent nowadays. Before coming to era of DL, some traditional ML techniques already have wide application across different industries. For examples, Random Forest[40], which combines the output of multiple decision trees, is utilized in Finance to evaluate customers with high credit risk[41], to detect fraud[42], and option pricing problems[43]. It also has applications within computational biology[44], facilitating doctors to tackle problems such as gene expression classification, biomarker discovery, and sequence annotation. DL comes into notice since it can deal with the high dimensional data, such as images, text, audio signals. One of biggest success of DL is the development of Convolutional neural network for image classification[45] and object recognition tasks[46]. CNNs have three main types of layers: convolutional layer, pooling layer and fully connected layer. The convolutional layer is the core building block of a CNN, which extract features from the images. Pooling layer reduce the dimension of data, help improve efficiency and limit the risk of overfitting. The fully connected layer is usually set at last to perform the task of classification based on the extracted features from previous layers. Recurrent neural network is another type of DNNs, dedicated to solve the sequential problems seen in natural language processing[47] and speech recognition[48]. Long short-term memory (LSTM) network is a popular RNN architecture, consisting of cells. Each cell have gates, long term and short term memory. These gates control the flow of information which is needed to predict the output in the network, while long term and short term memory are designed to solve the problem of long-term dependencies.

Despite the great potentials of ML techniques, ML, especially DL models, are suffering from a wide range of risks, such as the adversarial robustness, backdoor and transparency. This leads to the research area of verification and validation of DL (ref. to the comprehensive review on [49]). Verification provide the provable guarantee on decision safety of neural networks[16]. It can further analyse the reachability of neural networks[50]. Testing arises to complement the verification of DNNs, which is known to have the drawbacks of high computation complexity. A lot of testing techniques, such as coverage guided testing[51], distribution aware testing[52], fuzzing[53] are proposed to test the performance of DNNs. This thesis mainly focus on the testing category and intend to design new testing methods for different type of DNNs.

2.2 Test Machine Learning Component

2.2.1 Coverage Guided Testing

Traditional software testing and coverage methods cannot be easily applied to deep neural networks (DNNs). A number of new techniques for testing neural networks and measuring test coverage have been developed. The comparison of theses approaches are listed in Table 2.1.

Test Method	**Coverage Criteria**	**Test Generation**	**Distance Metric**	**DNNs Under Test**
DeepXplore[27]	Neuron Coverage	Joint Optimisation	L_1	Multiple
DeepTest[54]	Neuron Coverage	Greedy Search	Jaccard Distances	Single
DeepGauge[55]	Neuron Level Coverage, Layer Level Coverage	Adversarial Attack	L_∞	Single
DeepConcolic[51]	MC/DC, Neuron Coverage and its extensions	Concolic Testing	L_∞, L_0	Single
[56]	Quantitative k-Projection Coverage	0-1 Integer Programming	N/A	Single
SADL[57]	Surprise Coverage	Adversarial Attack	N/A	Single
TensorFuzz[58]	Approximate Nearest Neighbor	Fuzzing	N/A	Single
DeepStellar[4]	State-Level Coverage Transition-Level Coverage	Fuzzing	N/A	Single

Table 2.1: Comparison between different coverage-guided testing for DNNs

[27] firstly introduces the neuron coverage metric for the neural network using rectified linear units (ReLUs) as the activation functions in DeepXplore. According to the definition, a test suite is said to have achieved full coverage if there is at least one input to make each hidden unit in the neural network get a positive value. Then, they demonstrate that achieving high neuron coverage can find input which induce more differential behaviors of neural networks. The high coverage rate can be represented as the optimization problem and efficiently solved by gradient based search.

[54] offer a systematic approach, called DeepTest, for autonomously generating test scenarios that optimise neuron coverage in safety-critical DNN-based systems such as self-driving automobiles. They show empirically that variations in neuron coverage coincide with changes in the behaviour of an self-driving automobile. They further show how several realistic image transformation, such as changes in light and the weather conditions, may be utilised to generate synthetic tests that boost neuron coverage. They use transformation-specific metamorphic relations to identify incorrect behaviour automatically. Experiment

results also indicate that synthetic pictures may be utilised to retrain DNNs and improve the robustness to various corner situations.

[55] presents a set of new testing criteria based on multi-level and multi-granularity coverage for neural networks in DeepGauge, ranging from the coverage of different sections of neurons values to different combination of neuron activation patterns. The testing criteria can measure to what extent neuron's functionality is exercised and quantify defect detection ability of test data on neural networks. A higher coverage rate of testing criteria indicates a higher potential to detect more diversified neural networks' defects.

[51] adapts the Modified Condition/Decision Coverage (MC/DC), a conventional software coverage metric, to deep neural networks in DeepConcolic. MC/DC requires that all condition neurons should determine the outcome of the decision neuron independently. To efficiently increase the coverage rate, they develop the concolic testing by alternating between concrete execution and symbolic analysis to incrementally generate test cases.

[56] present quantitative k-projection coverage as a measure to control combinatorial explosion while directing the data sampling procedure. Assuming that domain experts in autonomous drivining offer independent environment conditions, such as weather, scenery, or partly obstructing pedestrians, and associating components in each condition with weights, the combination of these conditions produces scenarios, and the weights associated with each equivalence class may be interpreted as their relative importance. To achieve complete k-projection coverage, the data set, when projected to the hyperplane defined by arbitrarily chosen k-conditions, must cover each class with a minimum number of data points equal to the corresponding weight. The exact computation of k-projection coverage remains NP in the general situation when scenario construction is controlled by constraints. They propose theoretic complexity for essential sub-cases and an encoding to 0-1 integer programming in terms of determining the least number of test cases required to obtain complete coverage.

Surprise Adequacy for Deep Learning Systems (SADL), a unique test adequacy criteria for evaluating DL systems, is proposed by [57]. It is based on the behaviour of DL systems with regard to their training data. It quantifies an input's surprise as the difference in DL system behaviour between the input and the training data (i.e., what was learned during training), and then use this as an adequacy criterion: a good test input should be suitably but not overly unexpected when compared to training data. Empirical testing with a variety of DL systems ranging from simple image classifiers to autonomous driving car platforms demonstrates that systematic sampling of inputs based on their surprise can improve DL system classification accuracy against adversarial examples via retraining.

[58] develops the coverage-guided fuzzing framework, called TensorFuzz, where randomly mutated input are guided by the coverage metric, provided by approximate nearest neighbor (ANN) algorithms. Approximate nearest neighbor algorithms can decide if a new input should be added into the corpus based on building a data structure for finding sufficiently closed data points. They further combine the coverage-guided fuzzing with property-based testing, which automatically generates test cases that attempt to violate user-specified property, to detect numerous flaws in trained neural networks.

Few works contribute to the development of coverage metrics for RNNs. [4] take the first step toward quantitative study of RNN-based DL systems in DeepStellar. To describe RNN's internal characteristics, they model it as an abstract state transition system. They propose two trace similarity measures, based on state and transitions, respectively, and five coverage criteria, called Basic State Coverage, Weighted State Coverage, n-Step State Boundary Coverage, Basic Transition Coverage, and Weighted Transition Coverage, to allow for quantitative study of RNNs. They also present two methods for adversarial sample detection and coverage-guided test case generation that are driven by quantitative measurements. DeepStellar is thoroughly tested on four RNN-based systems and shown effective to capture the differences between samples even with very small perturbations and reveal erroneous behaviours of RNNs.

2.2.2 Distribution Aware Testing

There are increasing amount of DL testing works developed towards being distribution-aware. Deep generative models, such as Variational AutoEncoders (VAE) and Generative Adversarial Networks (GAN), are applied to approximate the training data distribution, since the inputs (like images) to Deep Neural Network are usually in a high dimensional space. Previous works heavily rely on Out of Distribution-Aware (OODA) detection [59, 52, 60] or synthesising new test cases directly from latent spaces [61, 62, 63], called Feature-Only Distribution-Aware (FODA). The comparison between two categories of works are listed in Fig 2.1.

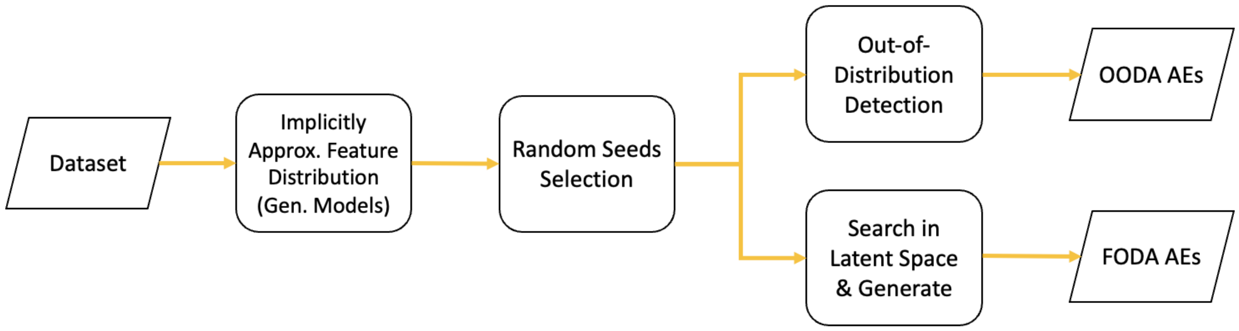

Dataset
Implicitly Approx. Feature Distribution (Gen. Models)
Random Seeds Selection
Out-of-Distribution Detection
OODA AEs
Search in Latent Space & Generate
FODA AEs

Figure 2.1: Comparison between OODA and FODA methods

[59] propose the first distribution-guided coverage criterion for generating unseed test cases, while giving a great assurance of the validity of the identified faults to DL system. To be specific, they integrate the OOD techniques into the coverage criterion. They obtain the OOD-score distribution from the training data using cutting-edge OOD algorithms. If a new test case has an OOD-score that exceeds a predefined threshold, it is labelled as OOD. The innovative OOD-guided coverage criteria may aid in filtering distribution-relevant errors. DNNs trained with distribution-relevant errors outperform those trained with errors that are unaware of the distribution. The findings highlight the significance of distribution knowledge and emphasise the need for caution while developing future DL testing frameworks.

[52] study the validity of test cases generated by existing DNN test generation techniques through variational auto-encoder (VAE). When compared to OOD inputs, a trained VAE model will produce high probability density estimates for data from the training data distribution. This critical insight is utilised to validate test inputs produced by DNN test generating approaches. They find that current coverage guided testing generates a huge number of invalid test cases, raising test costs without providing an obvious advantage. Furthermore, existing neuron-based coverage metrics for DNNs cannot differentiate between valid and invalid test cases, which risks biasing test suites toward including more invalid inputs in pursuit of better coverage. Based on these findings, they develop a novel approach that combines a VAE model with current test generation approaches to generate test cases that produce only valid inputs. They formulate the joint optimization of the probability density of valid inputs and the objective of current DNN test generation algorithms, and employ gradient ascent to generate valid test cases. The proposed technique is cost-effective, according to experimental study.

Test selection refers to the area of research focused on selecting a small set to label from a large set of unlabelled data, which are more representative to reveal errors in given DNN. [60] empirically study the test selection and retraining. They observe that retraining using both original training data and selected data achieves better model performance than using selected data only. In addition, even using the optimal retraining strategy, existing selection metrics show different performance under different data distributions. For example, random selection get the surprisingly best performance, when OOD data are more than 70% in the set (30% are in-distribution (ID) data). Moreover, class bias is another potential factor for data selection. Based on these observations, they further present a distribution-aware test (DAT) selection metric to reduce the impact of distribution shifts on model retraining. DAT's fundamental principle is to choose uncertain and representative data from the ID and OOD sets, respectively. To begin, they use an OOD detector to divide the new data into the ID set and the OOD set. Following that, DAT picks the most uncertain data for the ID set, which have the same distribution as the training data but have not been sufficiently learnt by the model. DAT chooses the best representative data for the OOD collection, which indicates that the data chosen may represent the whole set. According to the experimental results, DAT outperforms all other available test selection metrics.

In order to generate realistic, conformance to requirements, and error-revealing test cases, [61] leverage the variant of Conditional Variational Autoencoder (CVAE) to capture a manifold, representing the feature distribution of training data. CVAE can learn the manifold conditioned on the data label, and use encoder and decoder to map high dimensional data from or to the manifold. Once a CVAE has been trained, it is possible to sample new test cases from this manifold and map them to the original input space using the decoder. As a CVAE is tuned to produce such images with high probability, these test inputs are likely representative of distribution images. These test inputs may also be novel to the degree that a VAE can interpolate between existing data points, enabling the discovery of new issues utilising these inputs. The central aim of this study is to apply search-based test generation to the manifold space in order to generate novel, intriguing, and error-revealing test cases.

A fitness function is designed so that the uncertainty of the tested model is maximised, with the argument that high uncertainty inputs are more likely to cause errors.

[62] present a new method to assess the safety of DL system based on the concept frontier of behaviours. The frontier of behaviours of DL systems refer to a pair of inputs which are virtually similar to each other but induce different behaviors of DL system. The DL system is safe if the frontier of behaviours are outside the validity input domain. To search for frontier of behaviours, they propose the model-based input generation approach to generate the realistic input. For MNIST dataset, they convert the handwritten digit images to the Scalable Vector Graphics, which is combination of cubic and quadratic Bézier curves. The manipulation of the Bézier curve parameters can make sure the perturbed handwritten digits are realistic. The self-driving car is trained and tested on the BeamNG simulation environment. Such simulation system provides different scenarios, consisting the roads, the driving task, and environments, like weather conditions, and light. They further apply evolutionary algorithm to search for frontier of behaviours of DL systems.

[63] provide the first method, called distribution-based falsification and verification (DFV), that utilises environmental models to concentrate DNN falsification and verification on the meaningful input space. DFV automatically constructs an input distribution model using unsupervised learning, prefixes this model to the DNN to require all inputs from the learnt distribution, then reformulates the property to the input space of the distribution model. This transformed verification enables current DNN falsification and verification methods to focus on the input distribution, hence eliminating examination of inputs that are not meaningful. The investigation of DFV using seven falsification and verification tools, two DNNs defined over different data sets, and ninety-three distinct distribution models provides clear evidence that the counter-examples found by the tools are much more representative in the training data distribution.

2.2.3 Backdoor Testing

Detecting backdoor from a ML model is challenging, since only input containing backdoor trigger, which is only known by adversary, will induce the erroneous behaviors.

Test Method	**Detection Type**	**Methodology**	**Access**
[64]	Backdoor Input	Activation Clustering	White-box
Neural Cleanse [33]	Backdoor Trigger	Optimisation + Outlier Detection	White-box
NeuronInspect [65]	Backdoor Input	Output Explanations	White-box
DeepInspect [66]	Backdoor Trigger	Trigger Distribution Learning	Black-box
B3D [67]	Backdoor Trigger	Gradient-free Optimization	Black-box
AEVA [68]	Backdoor Trigger	adversarial extreme value analysis	Black-box hard-label

Table 2.2: Comparison between different backdoor detection for DNNs

The Activation Clustering (AC) approach is adopted for finding poisoned training samples designed to incorporate backdoors into DNNs [64]. This approach examines the neural network activations in the training data to see whether it has been poisoned and, if so, which datapoints are toxic. The idea behind this strategy is that although backdoor and target samples are classified the same by the compromised network, the rationale for this classification is different. The network identifies features in the input that it has learnt correlate to the target class in the case of standard samples from the target class. It identifies features associated with the backdoor trigger in the case of backdoor samples, causing it to identify the input as the target class. The network activations, which describe how the network makes its "decisions," should show this variation in methodology. The activations of last convolutional layer are obtained for all training data. They are grouped according to the label and each group is clustered separately. To cluster the activations, the dimensionality reduction technique, Independent Component Analysis (ICA) is applied. Then cluster analysis methods, like exclusionary reclassification, relative size comparison and silhouette score can help users identify the possible data poisoning from the clustered activations.

[33] develops the generalizable technique to detect and reverse engineer the backdoor trigger from DNNs, called "Neural Cleanse". The trigger reverse engineering process can be formalized as the optimization for the minimum perturbation to transform input label of any training data to a target class. Then, the Median Absolute Deviation based outlier detection algorithm is applied to identify the real trigger from the potential ones obtained from the optimization. In addition, they propose three methods of backdoor mitigation. The first one is the utilization of reverse engineered trigger to identify the backdoor related neurons. Then, the proactive filter can detect and filter out the suspected input which activate the backdoor related neurons. Second is pruning out the backdoor related neurons so that DNN model is patched. The last one is training DNN to unlearn the original trigger. The infected DNN will recognize the correct label even the backdoor trigger is present.

NeuronInspect [65] is a framework for discovering Trojan backdoors in deep neural networks using output explanation techniques. NeuronInspect identifies the presence of backdoor attack by creating an explanation heatmap of the output layer. The authors observe that heatmaps generated from clean versus backdoor embedded models exhibit distinct characteristics. Therefore, they extract features from an attacked model that quantify the sparse, smooth, and persistent characteristics of explanations. They combine these characteristics and employ outlier detection to identify the outliers, which constitute the attack targets. They demonstrate the efficacy and effectiveness of NeuronInspect using the MNIST digit recognition dataset and the GTSRB traffic sign recognition dataset. NeuronInspect outperforms the state-of-the-art trojan backdoor detection techniques, Neural Cleanse, by a significant margin in terms of robustness and efficacy when tested against a variety of attack scenarios.

DeepInspect [66] is the first practical backdoor detection framework that determines whether a DNN has been compromised ("sanity check of a pre-trained DNN model") with minimal information about the queried model. DeepInspect is comprised of three primary steps. Model inversion technique is firstly utilized to recover a substitution training dataset containing all classes. Then, a conditional Generative Adversarial Network is trained to generate the possible backdoor trigger with queried model as fixed discriminator. The queried model should predict the inversed sample added with trigger learned by generative model, as the target class. Finally, the perturbation level of recovered triggers is used as the test statistics for anomaly detection. DeepInspect's trigger generator enables model patching for effective backdoor mitigation. Extensive experiments demonstrate that DeepInspect provides superior detection performance and lower runtime overhead compared to previous work.

[67] propose a method for black-box backdoor detection (B3D). Similar to [33], B3D formulates backdoor detection as an optimization problem that is solved by reverse-engineering the potential trigger for each class using clean data. The main difference is that they employ a gradient-free algorithm that minimises the objective function solely through model queries to solve the problem. Moreover, they demonstrate the applicability of B3D when using synthetic samples when clean samples for optimization are not available. They conduct extensive experiments on multiple datasets to validate the efficacy of B3D and synthetic samples in detecting backdoor attacks on hundreds of DNN models, some of which are trained normally while others are backdoored. Due to the appropriate problem formulation and efficient optimization procedure, their methods achieve detection accuracy comparable to or even surpassing that of previous methods based on model gradients. In addition to detecting backdoors, they aim to mitigate any discovered backdoors in infected models. Due to the inability to modify the black-box model, the typical retraining and fine-tuning strategies cannot be implemented in a black-box environment. Therefore, they propose a simple yet effective strategy that rejects any input containing the trigger stamp in order to make accurate predictions without modifying the infected model.

Existing backdoor detection approaches often demand access to the poisoned training data, the training parameters of the DNNs, or the prediction confidence for each input sample, which are unavailable in many real-world applications, such as on-device deployed DNNs. [68] address the black-box hard-label backdoor detection issue, in which the DNN is completely black-box and only its final output label can be obtained. They tackle this problem from an optimization standpoint and demonstrate that the aim of backdoor detection is constrained by an adversarial objective. Further theoretical and empirical research reveals that this adversarial objective results in a solution with a highly skewed distribution; a singularity is likely to be found in the adversarial map of a backdoor sample, which they term the adversarial singularity phenomenon. Based on this fact, they propose the adversarial extreme value analysis (AEVA) to identify backdoor in black-box neural networks. AEVA relies on extreme value analysis of the adversarial map generated using monte-carlo gradient estimation. Extensive experiments spanning several common tasks and backdoor embedding indicate that their technique is successful in identifying backdoor embedding in black-box hard-label circumstances.

2.3 Evaluate Machine Learning Components

The evaluation of ML models can be categorised as the formal verification, which aims to find the safety radius, the estimation of Lipschitz constant and the statistical approaches. The robustness is the most popular property for evaluating ML models, while some research works focus on assessing ML models' prediction accuracy on the operational context.

[21] defines the concept of global robustness as the expectation of maximum safe radius, measured by L_0 norm for each input, over the test dataset. they show that evaluating the robustness in terms of L_0 norm distance is NP-hard and then present an approximation for computing lower and upper bounds on the network's robustness iteratively. The approach is anytime, in the sense that it returns intermediate bounds and robustness estimates that are gradually but strictly improved as the computation proceeds; tensor-based, in the sense that the computation is performed over a set of inputs simultaneously, rather than one by one, to enable efficient GPU computation; and has provable guarantees, in the sense that both the bounds and the robustness estimates can converge to the optimal values.

[69] presents a theoretical argument for transforming robustness analysis into a local Lipschitz constant estimation issue, and suggest using the Extreme Value Theory to evaluate it efficiently. CLEVER, which stands for Cross Lipschitz Extreme Value for nEtwork Robustness, is the result of study. The proposed CLEVER score is attack-independent and computationally practical for massive neural networks. Experiment results on various networks, including ResNet, Inception-v3, and MobileNet, show that CLEVER is consistent with the robustness indicator measured by the L_2 and L_∞ norms of adversarial examples from powerful attacks, and defended networks using defensive distillation or bounded ReLU achieve higher CLEVER scores. CLEVER is, as far as we know, the first attack-independent robustness measure that can be applied to any neural network model.

[19] develops a new metric to evaluate the local robustness of neural network based on the probability that property is violated under the input distribution. Instead of answering the binary question whether neural network is robust or not by verification, their approach provides the informative notion on how robust neural network model is. Since the occurrence of adversarial examples is rare event for well-trained neural network, they successfully adapt the multi-level splitting, an advanced Monte Carlo Sampling, to estimating the statistical robustness. The experiments confirm that their statistical evaluation of robustness can scale well to the large neural network model.

[70] focuses on the operational testing of DNNs. That is to determine DNN models' actual performance on the dataset collected from the operational context, the challenge of which arises from the high cost at labelling data. To reduce the number of labelled samples, they first adopt the Confidence-based Stratified Sampling to achieve efficient but fragile and limited to classifier, estimation. Then, they propose to leverage the representation learned by DNN in the last hidden layer to guide the sampling of unlabelled operational data. A small set of samples are selected by minimizing the cross-entropy with respect to the population. The experiments reveal that, as compared to simple random sampling, their strategy only takes around half as many labelled samples to obtain the same degree of accuracy.

[71] proposes a test selection technique (DeepEST) that actively searches for faulty test cases in a DNN's operational dataset, with the goal of assessing the DNN's expected accuracy by a small and "informative" test set (namely, with a large amount of misprediction samples) for follow-up DNN improvement. Experiments show that DeepEST gives precise DNN accuracy estimates, comparable to (and frequently greater than) conventional sampling-based

DNN testing approaches, while identifying up to 30 times more misprediction samples with the same test set size.

2.4 Safety Analysis of ML in Learning-Enabled Systems

Currently, most safety verification and validation work focuses on the ML components, including formal verification and coverage-guided testing. Research is sparse at the system level, and there is none (apparent) on state estimation systems. In [72], a compositional framework is developed for the falsification of temporal logic properties of cyber-physical systems with ML components. Their approaches are applied to an Automatic Emergency Braking System. A simulation based approach [73] is suggested to verify the barrier certificates – representing safety invariants – of autonomous driving systems with an SMT solver. In both papers, the interaction – or synchronisation between ML and other components is through a shared value, which is drastically different from the neural network enabled state estimation, where the synchronisation is closer to the message-passing regime.

In addition, in [74], a system with a sigmoid-based neural network as the controller is transformed into a hybrid system, on which the verification can be solved with existing tools. This approach may not generalise to general neural networks since it heavily relies on the fact that the sigmoid is the solution to a quadratic differential equation. In [75], a graybox testing approach is proposed for systems with learning based controllers, where a gradient based method is taken to search the input space. This approach is heuristic, and based on the assumption that the system is differentiable. The LESs cannot be verified with these approaches. Moreover, there is some early research on the robustness of Kalman filter by false information injection [76, 77], where the false information is modelled as Gaussian noise.

Chapter 3

Test DL Robustness through Coverage Metrics

3.1 Introduction

Feedforward neural networks (FNNs), notably convolutional neural networks (CNNs), are vulnerable in various safety and security scenarios, subject to adversarial attack [78], backdoor attack [79], data poisoning attack [80], privacy issues [81], etc. These defects are extensible to recurrent neural networks (RNNs). In this chapter, we study the RNN defects, mainly focusing on adversarial samples [82] (while our testing methods can be utilized to detect backdoor samples [83]). These defects will lead a well-trained RNN to mis-predictions. Different from CNNs, RNNs exhibit particular challenges, due to their more complex internal structures and their processing of sequential inputs with a temporal semantics, supported by their internal memory components. A generic RNN layer takes a sequential sample x as input, updates its internal state c, and generates an output h. Other structural components may be required for specific RNNs. Given an input $\{x_t\}_{t=1}^n$, the RNN layer will be unfolded with respect to the size n of the input, and therefore each structural component has a corresponding sequence of representations, for example $\{h_t\}_{t=1}^n$. Such a sequence of representations form a temporal evolution.

Coverage-guided testing has achieved a great success in software defect detection and extensively applied to FNNs. While existing coverage metrics for FNNs may be adapted to work with RNNs, they are insufficient because they do not work with the internal structures of RNNs and, more importantly, the most essential ingredient of RNNs – the temporal relation – is not considered. Moreover, we note that, a few coverage metrics are proposed in [4] for RNNs, by making simple extensions to those of FNNs without considering the temporal relation and the internal structures (e.g., the important components of RNNs such as gates). *This chapter is to develop dedicated coverage metrics for RNNs, to take into account the additional structures and temporal semantics.*

As suggested in [84, 85], a test metric does not have to be strongly related to adversarial samples, a specific type of defects corresponding to the robustness requirement of a neural network. This is not surprising, and actually not new (for software testing). As stated in [86], a (software) program with high test coverage has more of its source code executed during testing, which suggests it has a lower chance of containing undetected software defects compared to a program with low test coverage. We concur with this view, and suggest that, *instead of identifying a particular type of defects such as adversarial samples, coverage-guided testing is to generate a set of test cases as diversified as possible while preserving the naturalness, in order to exploit the internal behaviour of the neural networks that has real operational impact.* The proposed coverage metrics in this chapter are of such desirable features of being *diverse and natural* – with increased coverage, our approach is more likely to find different types of faulty behaviours (e.g., adversarial samples and backdoor samples) that manifest at multiple small regions in the input space (rather than adversarial samples clustered in one region as what normally attack-based methods find). Especially when the operational profile is unknown or changing, such diversified test cases are of particular importance for improving the delivered reliability [87] (indeed, spending all the budget on testing one input region that potentially has limited chance to be operated in practice is unwise). Meanwhile, our diversified test cases are "closer" to their seeds (points on the RNN's data manifold), compared to other state-of-the-art tools, implying higher chance to be seen in the real-life operation, thus preserving the naturalness.

Contributions We first discuss in Section 3.3 why the coverage-guided testing is useful in analysing RNNs and how to reasonably define the effectiveness of a testing framework. We focus on long short-term memory networks (LSTMs), which is the most important class of RNNs, and design three LSTM structural coverage metrics, namely boundary coverage (BC), step-wise coverage (SC) and temporal coverage (TC). Simply speaking, TC quantifies the multi-step temporal relation, which describes the internal behaviour on how LSTM cell processing inputs, while BC and SC quantify the value and single-step change of the temporal relation, respectively. We also discussed in Section 3.5 how to position the new metrics against a few closely related techniques such as complete verification techniques, existing metrics, etc.

We implement the proposed coverage metrics into a prototype tool TESTRNN[1], which includes two algorithms – a random mutation and a genetic algorithm based targeted mutation – for test case generation. In particular, targeted mutation uses the coverage knowledge to guide the test case generation. Initially, a random mutation is taken to generate test cases. Once the un-targeted randomisation has been hard to improve the coverage rate, a targeted mutation by considering the distance to the satisfaction of un-fulfilled test conditions is taken to generate corner test cases.

We conduct an extensive set of experiments over a wide range of LSTM benchmarks to confirm the utility of TESTRNN and the proposed coverage-guided RNN testing approach from the following aspects:

1. diversity of generated test cases (Section 3.7.2), with the observations that the LSTM

[1]https://github.com/TrustAI/testRNN

model's functional coverage can be approximated using our structural coverage metrics (Section 3.7.2) and our metrics complement existing metrics in guiding the exploitation of the input space (Section 3.7.2).

2. detecting defects (Section 3.7.3), with the observations that testRNN can not only find adversarial behaviours for the robustness of RNNs (Section 3.7.3) but also identify backdoor inputs for the security of RNNs (Section 3.7.3).

3. usefulness of test case generation (Section 3.7.4), with the observation that testRNN is efficient and effective in achieving high coverage rates (Section 3.7.4).

4. comparison with dedicated defect detection (Section 3.7.5), with the observation that our test method can find a set of more diversified adversarial samples, and these samples are more likely to occur in real world.

5. comparison with state-of-the-art tool DeepStellar (Section 3.7.6), with the observations that our metrics are better at guiding the exploitation of the input space and testRNN may achieve good coverage on the metrics in DeepStellar but not vice versa.

6. exhibition of LSTM internal working mechanism (Section 3.7.7), with the conclusion that semantic meanings behind the test metrics can help users understand the learning mechanism of LSTM model, making a step towards interpretable LSTM testing.

The organisation of the chapter is as follows. Section 3.2 gives the preliminaries. We will discuss the rationale of coverage-guided testing in Section 3.3. After this, we present our proposed test metrics in Section 3.4. This is followed by discussing in Section 3.5 how these new metrics are related to the formal-methods based verification techniques, existing coverage metrics, and adversarial defence techniques. We present our test case generation algorithm in Section 3.6 and the experimental evaluation in Section 3.7.

3.2 RNN Preliminaries

Feedforward neural networks (FNNs) model a function $\phi : X \to Y$ that maps from input domain X to output domain Y: given an input $x \in X$, it outputs the prediction $y \in Y$. For a sequence of inputs $x_1, \ldots, x_n$, an FNN ϕ considers each input individually, that is, $\phi(x_i)$ is independent from $\phi(x_{i+1})$.

By contrast, a recurrent neural network (RNN) processes an input sequence by iteratively taking inputs one by one. A recurrent layer can be modeled as a function $\psi : X' \times C \times Y' \to C \times Y'$ such that $\psi(x_t, c_{t-1}, h_{t-1}) = (c_t, h_t)$ for $t = 1...n$, where t denotes the t-th time step, c_t is the cell state used to represent the intermediate memory and h_t is the output of the t-th time step. More specifically, the recurrent layer takes three inputs: x_t at the current time step, the prior memory state c_{t-1} and the prior cell output h_{t-1}; consequently, it updates the current cell state c_t and outputs hidden state h_t.

RNNs differ from each other given their respective definitions, i.e., internal structures, of recurrent layer function ψ, of which long short-term memory (LSTM) in Equation (3.1) is the most popular and commonly used one.

$$\begin{array}{rcl} f_t & = & \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\ i_t & = & \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ c_t & = & f_t * c_{t-1} + i_t * \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \\ o_t & = & \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t & = & o_t * \tanh(c_t) \end{array} \tag{3.1}$$

In LSTM, σ is the sigmoid function and tanh is the hyperbolic tangent function; W and b represent the weight matrix and bias vector, respectively; f_t, i_t, o_t are internal gate variables of the cell. In general, the recurrent layer (or LSTM layer) is connected to non-recurrent layers such as fully connected layers so that the cell output propagates further. We denote the remaining layers with a function $\phi_2 : Y' \to Y$. Meanwhile, there can be feedforward layers connecting to the RNN layer, and we let it be another function $\phi_1 : X \to X'$. As a result, the RNN model that accepts a sequence of inputs $x_1, \ldots, x_n$ can be modeled as a function φ such that $\varphi(x_1 ... x_n) = \phi_2 \cdot \psi(\prod_{i=1}^{n} \phi_1(x_i))$.

Cell structure The processing of a sequential input $x = \{x_t\}_{t=1}^{n}$ with an LSTM layer function ψ, i.e., $\psi(x)$, can be characterised by gate activations $f = \{f_t\}_{t=0}^{n}$, $i = \{i_t\}_{t=0}^{n}$, $o = \{o_t\}_{t=0}^{n}$, cell states $c = \{c_t\}_{t=0}^{n}$, and outputs $h = \{h_t\}_{t=0}^{n}$. We let $\mathcal{S} = \{f, i, o, c, h\}$ be a set of structural components of LSTM, and use variable s to range over $\mathcal{S}$.

Sequential structure Each s represents one aspect of the concrete status of an LSTM cell. To capture the interactions between multiple LSTM steps, temporal semantics are often used to understand how LSTM performs [88]. Test metrics in this paper will rely on the structural information such as *aggregate knowledge* ξ_t^h and *remember rate* $\xi_t^{f,avg}$, as explained below, and their temporal relations.

Output h is seen as short-term memory (as opposed to c for long-term memory) of LSTM. It is often used to understand how information is updated, either positive or negative according to the value of h_t. Thus, we have

$$\begin{array}{rcl} \xi_t^{h,+} & = & \sum\{h_t(j) \mid j \in \{1, \ldots, |h_t|\}, h_t(j) > 0\} \\ \xi_t^{h,-} & = & \sum\{h_t(j) \mid j \in \{1, \ldots, |h_t|\}, h_t(j) < 0\} \\ \xi_t^{h} & = & |\xi_t^{h,+} + \xi_t^{h,-}| \end{array} \tag{3.2}$$

Where $|h_t|$ represents the size of vector h_t. Intuitively, ξ_t^h represents the *aggregate* knowledge regarding short-term memory.

The forget gate f is a key factor for long-term memory in LSTM, as it controls whether the aggregate information can be passed on to the next (unfolded) cell or not. The portion of information passed is then measured by $\xi_t^{f,avg}$ as follows.

$$\xi_t^{f,avg} = \frac{1}{|f_t|} \sum_{j=1}^{|f_t|} f_t(j) \tag{3.3}$$

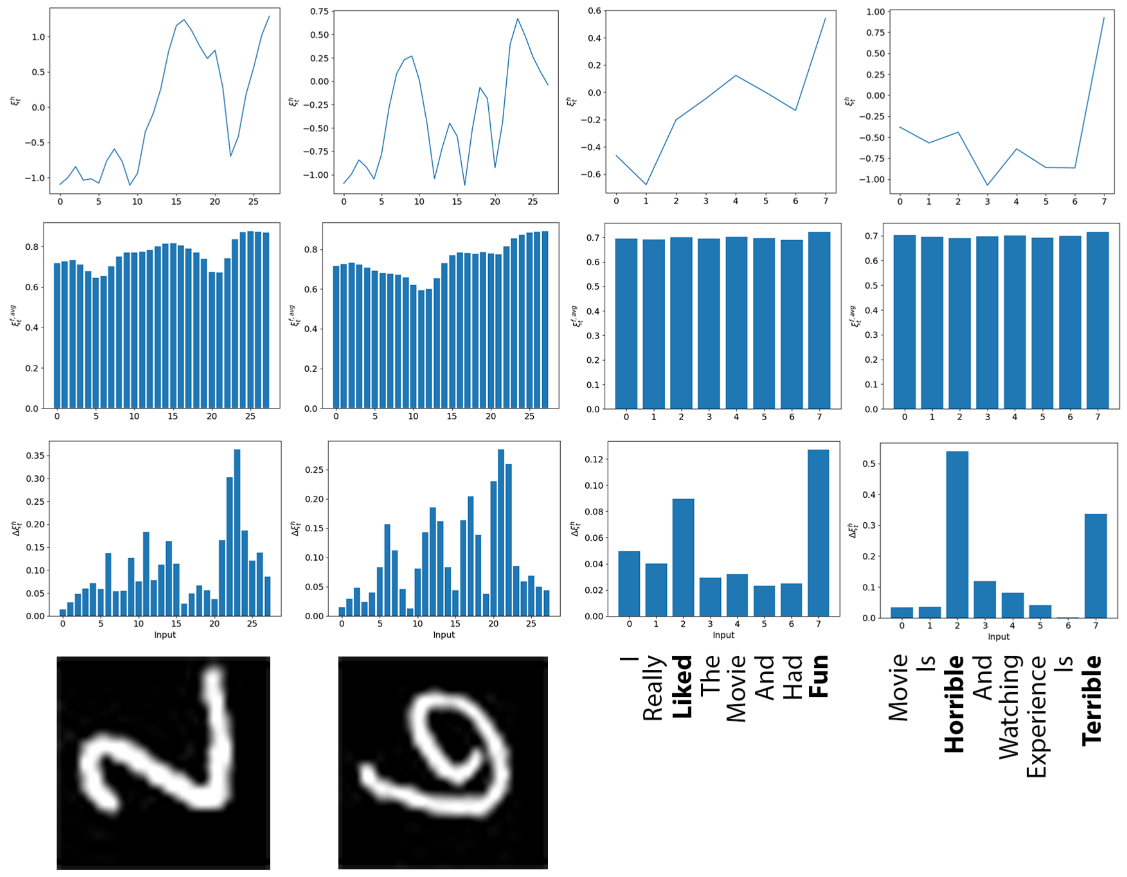


Figure 3.1: Examples to show how positive and negative elements of output vectors represent the information in MNIST and IMDB models. The x-axis includes the inputs (bottom row) and the y-axis includes $\mathcal{N}_z(\xi_t^h)$ (top row), $\mathcal{N}_m(\xi_t^{f,avg})$ (second row) and $\mathcal{N}_m(\Delta\xi_t^h)$ (third row) values. In MNIST, each column of pixels corresponds to a step in LSTM and in the IMDB model each step represents a word in the movie review.

Example 3.2.1. *Fig. 3.1 presents a set of visualisations to the temporal update of the abstract information. In particular, the top row contains curves for $\mathcal{N}_z(\xi_t^h)$ and the second row contains curves for $\mathcal{N}_m(\xi_t^{f,avg})$, changed with respect to the time. The third row visualises the evolution of step-wise change information $\mathcal{N}_m(\Delta\xi_t^h)$. $\mathcal{N}_z$ and $\mathcal{N}_m$ are two normalization function which will be introduced later.*

Let $\mathcal{A} = \{+, -, avg\}$ be a set of symbols representing the abstraction functions as in Eq. (3.2-3.3). The above can be generalised to work with any $s \in \mathcal{S}$ and $a \in \mathcal{A}$. For $\xi_t^{s,a}$, once given a fixed input x, we may write $\xi_{t,x}^{s,a}$.

3.3 Problem Statement

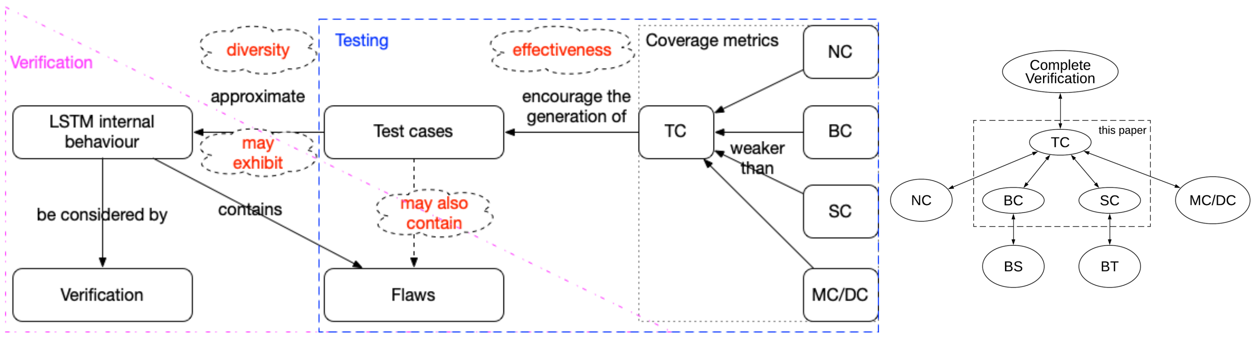


Figure 3.2: (Left) Connection of Verification and Coverage Guided Testing Frameworks. Verification and testing overlap on "Flaws", representing that they have the same objective. From verification to testing, an approximation is made, i.e., test cases approximate the LSTM internal behaviour. Guidelines (colored with red) are needed to ensure the approximation quality. (Right) Relation between Coverage Metrics. NC: neuron coverage [3], BS: basic state coverage [4], BT: basic transition coverage [4], MC/DC: modified condition/decision coverage [5]. Arrows represent the "weaker than" relation between metrics.

This section explains why the coverage guided testing is useful in analysing RNNs and how to reasonably justify its effectiveness. Fig. 3.2(Left) presents the connection between verification and testing in this context. While incomplete, testing has been shown practical – and in many cases sufficiently effective – in providing assurance to the quality of software. As in Fig. 3.2, these two approaches overlap on the "Flaws". Verification can detect defects because it exhaustively exploits all internal behaviour of RNNs, which include defective behaviours. On the other hand, testing approach uses test cases to approximate – or sample – the internal behaviour. Due to the finite sampling, defects may or may not be detected. Therefore, testing needs to be systematic to be effective in defect detection, and coverage-guided testing is one of the main approaches.

Due to the size and the temporal semantics of RNNs, it becomes important to find a (meaningful) set of metrics to guide the sampling or the test case generation. Our proposed set of coverage metrics plays the role of such guidance – as suggested in Fig. 3.2(Left), coverage-guided testing generates a set of test cases to exploit the internal behaviour of the neural networks. We note, such coverage metric does not have to be strongly correlated to adversarial samples [84, 85] (a specific type of defects corresponding to the robustness requirement of a neural network). What really matters is, within the testing budget, to find defects that are as diverse and natural as possible so that fixing them would gain maximised impact on the delivered reliability.

There are two main goals of testing [89]: debug testing, which probes the software for defects, and operational testing, which is to gain confidence that the software is reliable [90]. The former seeks test cases to excite as many failures as possible (then we may fix the defects behind them), but the test cases normally are not representative of the software's day-to-day operation. Confidence in the delivered reliability can only be gained by the later, i.e. testing that represents the typical usage (the operational profile) [91]. Coverage-guided testing belongs to the former, while our method should also be designed in the best interest for the operational reliability. That is, our test cases should be more likely generated from the high probability density area on the operational profile, compared to attack-based and other state-of-the-art debug testing methods, so that the defects found are more "practical" in the sense that fixing them would effectively improve the operational reliability.

The question is on how effective a specific testing approach is when exploiting the internal behaviour. Below, we provide a few guidelines by evaluating the connections of entities in Fig. 3.2(Left) (shown with red color).

First, the test cases are required to be *diversified and natural*, so as to cover the LSTM internal behaviour comprehensively. This is to avoid the test cases being clotted together in a small region of the input space (representing a certain type of defects) and lose the ability of finding other defects manifested in different regions. However, it is easy to generate diversified test cases by "forcing diversity" (e.g., by selecting inputs that maximise the average inter-point distance). Thus diversity criteria is only sensible when paired with naturalness – tests cases should not be far from the RNN's data manifold (i.e., potentially high density area of its future optional profile). Second, the test cases can reveal defects. While it is hard to establish strong correlation between test metrics and a specific type of defects, it is still a reasonable request that the generated test cases *reveal* defects as many as possible. Third, the test case generation algorithm needs to be effective, in terms of its ability in improving the coverage rate with diversified and natural test cases. Finally, to show that the generated test cases are sufficiently representative, we may use the test cases to exhibit the working mechanism of LSTM.

Moreover, given the complexity of the internal behaviour in RNNs, we believe a set of coverage metrics are needed to ensure that the above guidelines can be achieved. In an ideal case where the testing budget is sufficient, our metrics in the paper may complement – instead of replace – others, and vice versa. These guidelines will be used when designing our experiments in Section 3.7.

3.4 LSTM Test Coverage Metrics

In this section, we present a family of three coverage metrics (BC, SC and TC) for the testing of LSTM models. These metrics take into account both the values of structural information $\xi_t^{s,a}$ for $s \in \mathcal{S}$ and $a \in \mathcal{A}$ as in Eq. (3.2-3.3) and their step-wise and bounded-length temporal relations. We utilize two normalization methods for the convenience of determining thresholds, independent of specific dataset. $\mathcal{N}_z$ and $\mathcal{N}_m$ are z-score and min-max normalization function defined as below:

$$\mathcal{N}_z(\xi) = \frac{\xi - \mu}{\sigma}, \quad \mathcal{N}_m(\xi) = \frac{\xi - min}{max - min}$$

where parameters μ, σ, min and max can be calculated based on the statistics of ξ in training dataset. The z-score normalization is suitable for preprocessing test conditions quantifying the relations between different features, e.g., TC, while min-max normalization is better for the test conditions to hit the large activation values, e.g., BC and SC. Given a time series of length n, we can choose the sequence of interest $[t_1, t_2]$ ($t_1 \geq 1$ and $t_2 \leq n$) to implement the following coverage metrics.

Boundary Coverage (BC) Boundary values are often regarded as important cases in software testing, as they could exploit extreme software behaviours. We therefore define BC for depicting test conditions that cover the boundary values of the LSTM data flow as follows.

$$\{\mathcal{N}_m(\xi_t^{s,a}) \geq \alpha_{max},\ \mathcal{N}_m(\xi_t^{s,a}) \leq \alpha_{min} \mid t \in \{t_1 ... t_2\}\}$$

Thresholds α_{max} and α_{min} are chosen from interval $[0, 1]$. The min, max values can be estimated using values computed over the training dataset $\{\xi_{t,x}^{s,a} \mid t \in \{t_1 ... t_2\}, x \in \mathcal{D}_{train}\}$.

Example 3.4.1. *Suppose that there is a test condition $\mathcal{N}_m(\xi_t^{i,avg}) > 0.9$. It requires that the $\mathcal{N}_m(\xi_t^{i,avg})$ value is greater than threshold 0.9. Intuitively, this condition exercises LSTM's learning ability on the input at time t. As Eq. 3.1 shows, the input gate i controls how much information from the input is received by the network: $\mathcal{N}_m(\xi_t^{i,avg}) = 1$ implies that all its information is added to the long-term memory c and $\mathcal{N}_m(\xi_t^{i,avg}) = 0$ implies that no input information is added.*

Step-wise Coverage (SC) SC characterizes the temporal changes between connected cells. We use $\Delta\xi_t^s = |\xi_t^{s,+} - \xi_{t-1}^{s,+}| + |\xi_t^{s,-} - \xi_{t-1}^{s,-}|$ to outline the maximum change of the structural component $s \in \mathcal{S}$ at time t. E.g., $\Delta\xi_t^h$ is the change of short-term memory at time t. Then, the SC test conditions are defined as follows.

$$\{\mathcal{N}_m(\Delta\xi_t^s) \geq \alpha_{SC} \mid t \in \{t_1 ... t_2\}\}$$

This set defines test conditions for LSTM's step-wise updates that exceed a threshold α_{SC}. Parameters for $\mathcal{N}_m$ are derived from $\{\Delta\xi_{t,x}^s \mid t \in \{t_1 ... t_2\}, x \in \mathcal{D}_{train}\}$.

Example 3.4.2. *The intuition behind step-wise coverage is to capture these significant inputs to the LSTM. As shown by the sentiment analysis LSTM example in Fig. 3.1 (Right), given two inputs, sensitive words "like", "horrible", "fun" trigger greater $\mathcal{N}_m(\Delta\xi_t^h)$ values than words "movie", "really", and "had".*

Temporal Coverage (TC) While the power of LSTM comes from its ability to memorize values over arbitrary time intervals, its test metrics need to ensure that the temporal patterns of memory updates are fully tested. This is essentially a time series classification problem and is intractable. In this part, we define test conditions to exploit temporal patterns of bounded length. Different from dynamic systems where the temporal relation can be infinite [92], the temporal relations in RNNs are always finite, because of the finite-sized input. Therefore, the bounded length does not lower the expressiveness of the test conditions. In particular, to facilitate the enumeration of all test conditions, we refer to symbolic aggregate approximation (SAX) [93] to convert any complicated time series of length v (v is usually a large number) into a symbolic sequence of length w ($v >> w$).

First of all, given any temporal curve $\xi^s = \{\xi^s_t\}_{t=t_1}^{t_2}$, we can reduce the dimension of temporal sequence from $v = t_2 - t_1$ to w following Piece-wise Aggregate Approximation (PAA).

$$\hat{\xi}^s_j = \frac{w}{v} \sum_{t=\frac{v}{w}(j-1)+t_1}^{\frac{v}{w}j+t_1} \xi^s_t \tag{3.4}$$

The main idea of PAA is to approximate the original time series by splitting them into w equal sized segments and average the values in each segment. For example, the temporal curve in Fig. 3.3 is split into $w = 5$ dimensions. The new approximated curve is denoted as $\hat{\xi}^s = \{\hat{\xi}^s_j\}_{j=1}^{w}$.

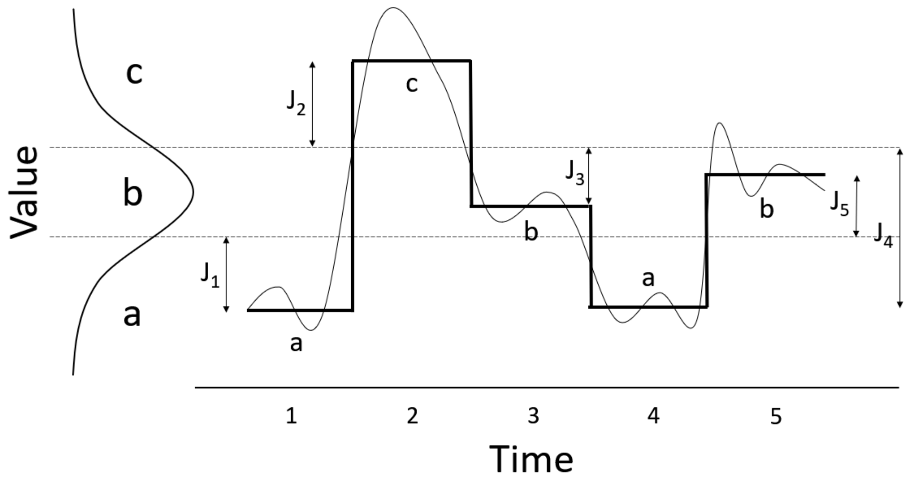


Figure 3.3: Illustration of projecting a temporal curve (Gaussian distribution) into a sequence of symbols *acbab*.

Then, we can define the symbolic representation for the temporal curve after dimensionality reduction. We start from z-normalizing $\hat{\xi}^s$ and discretising $D(\mathcal{N}_z(\hat{\xi}^s))$ – the domain of $\mathcal{N}_z(\hat{\xi}^s)$ – into a set Γ of sub-ranges. This discretization can refer to the distribution of $\mathcal{N}_z(\hat{\xi}^s_j)$, which can be estimated by conducting probability distribution fitting over the training dataset (since $\hat{\xi}^s$ is z-normalized, $\mathcal{N}_z(\hat{\xi}^s_j)$ is subject to the standard normal distribution).

Then, every normalized time series $\{\mathcal{N}_z(\hat{\xi}_j^s)\}_{j=1}^w$ can be represented as a sequence of symbols in the standard way. For example, in Fig. 3.3, the continuous space of $\mathcal{N}_z(\hat{\xi}^{s,a})$ is split into a set of three sub-ranges $\Gamma = \{a, b, c\}$.

Finally, test conditions from TC for covering a set of symbolic representations across multiple time steps $[t_1, t_2] \subseteq [1, n]$ can be expressed as follows.

$$\{\ell_1 \ell_2 ... \ell_w \mid \ell_j \in \Gamma, j \in [1, w]\} \tag{3.5}$$

Essentially, TC requests the testing to meet a set of temporal patterns for a specific time span $[t_1, t_2]$. The total number of temporal patterns for TC to cover is $|\Gamma|^w$. We remark that with the help of SAX, test conditions in TC is scalable in tackling the complexity of time series.

Example 3.4.3. *Fig. 3.1 (top row) demonstrates the temporal curve of hidden memory for each input across a selected time span. The curve is for $\mathcal{N}_z(\xi_t^h)$ and it is a clear illustration on the information processing of LSTM for each input. Fig. 3.3 further shows how a time series is converted into its symbolic representation acbab with $\Gamma = \{a, b, c\}$ and $w = 5$.*

3.5 Relation with RNN Defects

The aforementioned three coverage metrics encourage the exploration of the LSTM internal behaviour, which is helpful in detecting the RNN defects. In this section, we discuss the rationale behind by referring to the general relation (Fig. 3.2(Right)) between our new coverage metrics, the complete verification and a few existing coverage metrics. All discussions are supported by our experiments in Section 3.7.

Comparing with Formal Methods-based Verification Techniques For an input of length n, we denote the collection of curves (as in Fig. 3.3) for f, i, o as $Curv$. It precisely identifies an output (extracted features of the LSTM layer) and corresponds with a set of inputs (which cannot be differentiated by the LSTM layer). Let C_f, C_o, and C_i be the (possibly infinite) set of possible curves for their respective gates f, o, i. We have $Curv = C_f \times C_o \times C_i$. We also have curves C_h and C_o, which can be obtained from $Curv$ according to Equation (3.1).

The formal methods-based verification determines if there is an input that can lead to any unexpected behaviour. That is, it is equivalent to determine if there is a combined curve in $Curv$ that leads to the unexpected classification[2]. To this end, TC (and its test case generation) can be seen as an approach to exhaustively, but discretely, explore one of the curve sets C_s for $s \in \mathcal{S}$. Therefore, while the combination of TC working on different gates may provide a complete verification, in general the exploration of internal behaviour through TC is a necessary, but insufficient, approach for verification.

[2]As shown in the experiments, an unexpected classification can be normal mis-classification or caused by e.g. backdoor attacks and adversarial inputs.

Comparing with Other Coverage Metrics The purpose of TC is to encourage the exploration of either C_f, C_o, or C_i with the generated test cases. However, such exploration may be computational intensive – the complexity is exponential with respect to the length n. Since BC and SC do not consider the temporal relation between steps or boundary values, they are computationally more manageable. Assuming that under ideal parameter settings (e.g., thresholds α_{max} and α_{SC} and the set Γ of symbols), we say that a metric A is weaker than another B if for any test suite, it cannot have a lower coverage rate w.r.t. A than that of B. It is not hard to see that, both BC and SC are weaker than TC, and BC and SC are incomparable, as shown in Fig. 3.2(Right).

Besides the new BC and SC, TC is stronger than existing coverage metrics that are originally proposed for CNNs. For example, neuron coverage (NC) [3], which requires the coverage of neurons whose value is over a threshold (e.g., 0 for ReLU activation function), can be adapted to work with say the gate value or hidden state value. In this case, it is weaker than TC and incomparable with SC. Although NC and BC have similar formal expressions, BC concerns the boundary value rather than a value that indicates the activation status. For the MC/DC metrics [5], they can be adapted to work between time steps in the new context of RNNs. With such adaptation, MC/DC encourages the exploration of relations between time steps, and therefore are weaker than TC.

DeepStellar [4] abstracts the evolution of hidden states of an RNN into a discrete-time Markov chain (DTMC) before considering state and transition coverage. Its basic state coverage (BS) and basic transition coverage (BT) are designed to cover the possible state values and possible transitions. Given that the DTMC is an abstraction of the curves C_h, BS and BT are both weaker than BC and SC, respectively.

Remarkably, the relations in Fig. 3.2(Right) are based on theoretical analysis under "ideal parameter settings". They do not hold for any parameter settings. In our experiments in Section 3.7, we might observe the coverage rates on the same test set and the aforementioned relations are not in alignment, because of the specific parameter settings used (cf. Table 3.2).

Defence Techniques for Robustness and Security Some effective adversarial defence techniques, e.g. [94], are based on the observation that the adversarial samples exhibit different internal behaviour to those behaviour of training data samples. For security concerns like backdoor attack, activation patterns are also considered in the detection techniques such as [95]. Consequently, with the exploration of more RNN internal behaviour other than those appeared in the training data, it is more likely that RNNs defects will be exposed.

3.6 Coverage Guided Test Case Generation

Coverage metrics in Section 3.4 define test conditions that request particular patterns of long/short-term updates of abstracted information across multiple LSTM time steps. Given an LSTM network and a specific test metric, the **coverage rate** denotes the percentage of test conditions that have been satisfied over a set of test cases, i.e., test suite. To more efficiently achieve high coverage rate, in this section, we develop the coverage guided test case generation, as outlined in Fig. 3.4. We remark that, although focus of this paper is LSTM, the proposed testing approach (including both test metrics and tests generation) can be extended to work with other kinds of RNNs which use customized recurrent layer structures.

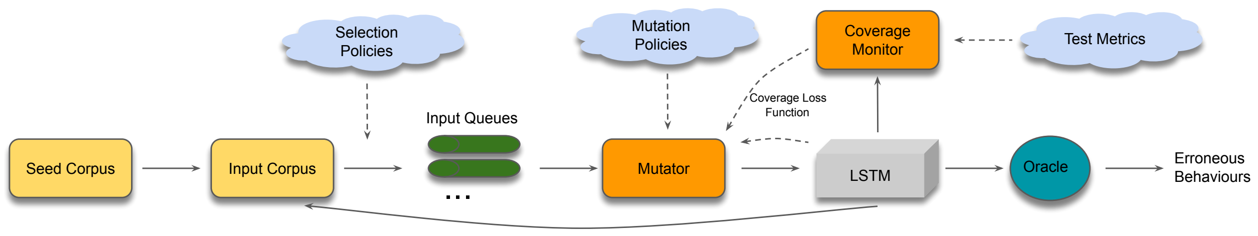


Figure 3.4: Coverage-guided LSTM testing in TESTRNN

The TESTRNN test case generation algorithm is detailed in Alg. 1. The test suite $\mathcal{T}$ is initialized with $\mathcal{T}_0$, a corpus of seed inputs (Line 1). New test cases are generated by mutating seed inputs. It keeps the traceability of test cases via a mapping *orig* that maps each test case generated back to its seed origin.

The main body of Alg. 1 is a loop (Lines 5−10) that iterates unless some target coverage level is reached (Line 4). At each iteration, a test input x is selected from the input corpus $\mathcal{T}$ (Line 5), and it is mutated following the pre-defined mutation function m (Line 6). Newly generated test inputs are added into $\mathcal{T}$ (Line 7), where they are queued for the next iteration. If the generated test case does not pass the oracle (Section 3.6.3), it represents a defect and it is added to $\mathcal{T}_{adv}$ (Lines 9-10).

3.6.1 Selection Policies and Queuing

Not all inputs in the corpus $\mathcal{T}$ are equivalently important, and they are ranked once added to the input queue (as illustrated in Fig. 3.4). When sorting queuing inputs on $\mathcal{T}$ for the Mutator engine, TESTRNN particularly prioritizes two kinds of test inputs: those that are promising in leading to the satisfaction of un-fulfilled test conditions and those that can trigger erroneous behaviours.

Thanks to its modular design, new selection policies can be easily integrated into TESTRNN as plug-ins (as indicated by cloud shapes in Fig. 3.4). The design of TESTRNN also features its high parallelism. The use of dynamically allocated input queues further optimises its runtime performance.

3.6.2 Mutation Policies

The Mutator engine lays at the core of TESTRNN. In particular, there are two types of mutation function m in Alg. 1: random mutation m_{rnd} and targeted mutation m_{targ}.

Algorithm 1: TESTRNN Algorithm

```
Input:
$\phi$: RNN to be tested
$\mathcal{T}_0$: a set of seed inputs
$m$: a mutation function
$r_{oracle}$: oracle radius
Output:
$\mathcal{T}$: a set of test cases
$\mathcal{T}_{adv}$: a set of discovered adversarial samples
1  $\mathcal{T} \leftarrow \mathcal{T}_0$
2  $orig \leftarrow dict()$
3  $orig[x] \leftarrow x$ for all $x \in \mathcal{T}_0$
4  while coverage rate is not satisfied do
5      $x \leftarrow$ select an element from $\mathcal{T}$
6      $x' \leftarrow m(x)$
7      $\mathcal{T} \leftarrow \mathcal{T} \cup \{x'\}$
8      $orig[x'] \leftarrow orig[x]$
9      if $||orig(x) - x'||_2 \leq r_{oracle}$ and $\phi(x) \neq \phi(x')$ then
10         $\mathcal{T}_{adv} \leftarrow \mathcal{T}_{adv} \cup \{x'\}$
11 return $\mathcal{T}, \mathcal{T}_{adv}$
```

Random Mutation When the LSTM input has continuous values (e.g., image input), Gaussian noises with fixed mean and variance are added to the input. Meanwhile, for discrete value input (e.g., IMDB movie reviews for sentiment analysis), a set of problem-specific mutation functions $\mathcal{M}$ are defined. The detail is in the experiment set-up (Section 3.7.1).

Targeted Mutation The targeted mutation is based on genetic algorithm for test case generation. Genetic algorithm is an evolutionary approach inspired by the process of natural selection. Mutations are selected only when they improve over the existing test cases on some pre-defined fitness function. The implementation of genetic algorithm comprises of four steps: initialization, selection, crossover and mutation. The last three steps are running iteratively till the solution is found.

Initialization. Firstly, we initialize the population by choosing a test case from the previous running cases. The test case is very close to the satisfaction of test condition.

Selection. Next, we select best few test cases from the population to the mating pool, evaluated by the fitness function. For the three classes of test conditions (BC, SC, TC) with respect to some $s \in \mathcal{S}$ and $a \in \mathcal{A}$, we define the following fitness function as the distance to their respective targets, e.g.,

$$\begin{aligned}J_{BC}(x) &= \alpha_{max} - \mathcal{N}_m(\xi^{s,a}_{x,t})\\ J_{SC}(x) &= \alpha_{SC} - \mathcal{N}_m(\Delta\xi^{s}_{x,t})\\ J_{TC}(x) &= \sum_{j=1}^{w} dist(\mathcal{N}_z(\hat{\xi}^{s}_{x,j}), u_j)\end{aligned}$$

where t, t_1, t_2, j are time steps that can be inferred from the context. $u_j = [u_l, u_r]$ is the interval of sub-range, represented by some symbol in Γ. The fitness of temporal curve to the targeted symbolic curve is to calculate the Manhattan distance, the absolute difference between structure value $\mathcal{N}_z(\hat{\xi}^{s}_{x,j})$ and the symbolic interval u_j is

$$dist(\mathcal{N}_z(\hat{\xi}^{s}_{x,j}), u_j) = \begin{cases} \mathcal{N}_z(\hat{\xi}^{s}_{x,j}) - u_r & \text{if } \mathcal{N}_z(\hat{\xi}^{s}_{x,j}) > u_r \\ u_l - \mathcal{N}_z(\hat{\xi}^{s}_{x,j}) & \text{if } \mathcal{N}_z(\hat{\xi}^{s}_{x,j}) < u_l \\ 0 & \text{else} \end{cases}$$

Intuitively, the fitness function (also called coverage loss) $J(x)$ is estimates the distance to the satisfaction of an un-fulfilled test condition. $J(x) \leq 0$ means that the test condition is covered. By generating test cases with the objective of gradually minimising the loss, the targeted mutation is essentially a greedy search algorithm.

Example 3.6.1. *In Fig. 3.3, the symbolic representation of temporal curve is acbab; the fitness of the curve to test condition bbcca can be calculated as* $J_{TC} = J_1 + J_2 + J_3 + J_4 + J_5$.

Crossover. Crossover is trivial in our test case generation for RNNs. This is mainly because the inputs to RNNs are usually discrete, which means the offspring of two parents by crossover (like exchanging chromosome) may be invalid due to the undefined semantic meanings. To avoid the validity issue, we skip this step and using mutation methods directly.

Mutation. We randomly mutate the test cases in the mating pool with user-defined function $\mathcal{M}$ in order to generate a new population for the next iteration. The previous population is replaced with the new one. It should be noticed that the parents in the mating pool are also added to the new population to make sure that solutions are towards good directions during the iterations.

The targeted mutation m_{targ} is shown in Algorithm 2. At each iteration, we choose k (or $|P|$, whichever is smaller) best individuals in the population P. Each individual is utilized to generate n test cases via the random mutation function m_{rnd}. The old population P will be replaced with the mutants along with their k parents in P^*. The whole process is repeated until the test condition is met or the maximum iteration is exceeded.

3.6.3 Test Set Evaluation

Test Oracle Test oracle determines if a test case passes or fails. We define a set of norm-balls, each of which is centered around a data sample with known label. The radius r_{oracle}

Algorithm 2: Targeted Mutation

```
Input:
$J$: fitness function
$\gamma$: maximun iteration number
$k$: number of parents for mating pool
$n$: number of offsprings mutated from one parent
$m_{rnd}$: a random mutation function
$x'$: a test case that is the closest to satisfy test condition
Output:
$x'_{new}$: a test case covering new test condition
1 $itr \leftarrow 0$
2 $P \leftarrow \{x'\}$
3 while test condition is not satisfied in $P$ and $itr < \gamma$ do
4     sort individual $x \in P$ according to fitness $J(x)$
5     $P^* \leftarrow$ highest sorted $\min\{k, |P|\}$ individuals in $P$
6     $P \leftarrow P^* \cup m_{rnd}(P^*, n)$
7     $itr \leftarrow itr + 1$
8 $x'_{new} \leftarrow \text{argmin}_{x \in P} J(x)$
9 return $x'_{new}$
```

of norm-balls intuitively means that a human cannot differentiate between inputs within a norm ball. In this paper, Euclidean distance, i.e. L^2-norm $||\cdot||_2$ is used. A test case x' is said to not pass the oracle if (1) x' is within the norm-ball of some known sample x, i.e., $||x - x'||_2 \leq r_{oracle}$, and (2) x' has a different classification from x, i.e., $\varphi(x) \neq \varphi(x')$. Take the definition in [96], a test case does not pass the oracle is an adversarial sample. We use **adversary rate** to denote the percentage of test cases that do not pass the oracle.

Diversity of Test Set More diversified test cases will explore more input space and thus are more likely to uncover different defects. Unfortunately, a unified, accurate way to measure diversity may not exist. We consider the following three intuitive, yet measurable proxies to the diversity.

First, diversity can refer to the number of categories the generated test cases belonging to. Intuitively, if the labels of two test cases are different, they are dissimilar and more diversified than two test cases with the same label. Second, test metrics (e.g., neuron coverage, SC, BC and TC) are to guide the exploitation of different internal behaviours of RNNs (cf. Section 3.4). Therefore, a test set that can achieve higher coverage on more test metrics is more diversified than the other. Third, if the distance in input space, measured with L_1, L_2, and L_∞ norm, can represent the semantic similarity between test cases, we may define the diversity by quantifying the relative positions of test cases to the seed input. Suppose a test set $\mathcal{T}$ contains n test cases, generated from a seed x_0, the angular-based diversity measure

[97] of $\mathcal{T}$ is

$$Diversity(\mathcal{T}) = -(\sum_{i,j=1}^{n} \frac{< x_i - x_0, x_j - x_0 >}{||x_i - x_0|| \; ||x_j - x_0||})/n^2 \tag{3.6}$$

This diversity measure is formed by the cosine similarity [98] and bounded by $[-1, 1]$. Since the test cases are generated by adding small perturbations to the seed input, the angular-based diversity is to measure if the test cases are uniformly distributed around the seed input x_0. A larger $Diversity(\mathcal{T})$ represents a more diversified $\mathcal{T}$.

3.7 Evaluation

We evaluate our testRNN approach with an extensive set of experiments from the following aspects: (1) the diversity of test cases generated under the guidance of the coverage metrics (Section 3.7.2), (2) the ability of detecting RNN defects (Section 3.7.3), (3) the effectiveness of the test case generation algorithms (Section 3.7.4), (4) the advantages over state-of-the-art attack tool [99] (Section 3.7.5), (5) the difference from state-of-the-art RNN testing tool DeepStellar [4] (Section 3.7.6), and (6) the exhibition of LSTM internal working mechanism (Section 3.7.7). Specifically, we study the following research questions (RQs):

- **RQ1**: will the exploitation of internal behaviour lead to the testing of different LSTM functions?
- **RQ2**: are our new metrics needed when we already have existing metrics?
- **RQ3**: will the exploitation of internal behaviour lead to the detection of adversarial samples?
- **RQ4**: will the exploitation of internal behaviour lead to the identification of backdoor attacks?
- **RQ5**: Can the test case generation algorithm achieve high coverage for the proposed test metrics?
- **RQ6**: What are the advantages of testRNN over attack-based methods for detecting adversarial samples?
- **RQ7**: What are the similarities and differences between DeepStellar and testRNN?
- **RQ8**: Are the testing results based on the proposed test metrics helpful on making LSTM interpretable?

All the experiments are run on a desktop with Intel(R) Core(TM) i7 CPU @ 3.80 GHz and 16 GB Memory.

3.7.1 Experimental Setup

RNNs under Evaluation

Our experiments are conducted on a diverse set of LSTM benchmarks, including:

MNIST Handwritten Digits Analysis by LSTM The MNIST database, containing a set of $60,000$ grey-scale images of size 28×28, is used to train a RNN model with 4 layers. The first two layers are LSTM layers, which are correspondingly connected and fed with rows of input images. That is, each input image is encoded as the row vector of shape $(28, 128)$ by the first LSTM layer, and then second layer will do further processing to output an image vector representing the whole image. Finally, two fully-connected layers with ReLU and SoftMax activation functions respectively, are used to process the extracted feature information to get the classification result. The model achieves 99.2% accuracy in training dataset ($50,000$ samples) and 98.7% accuracy in the default MNIST test dataset ($10,000$ samples).

Sentiment Analysis by LSTM The sentiment analysis network has three layers, i.e., an embedding layer, an LSTM layer, and a fully-connected layer, with 213301 trainable parameters. The embedding layer takes as input a vector of length 500 and outputs a 500×32 matrix, which is then fed into the LSTM layer. Subsequently, there is a fully-connected layer of 100 neurons.

Lipophilicity Analysis by LSTM We trained an LSTM regression network on a Lipophilicity dataset from the MoleculeNet [100]. The model has four layers: an embedding layer, an LSTM layer, a dropout layer, and a fully connected layer. The input is a SMILES string representing a molecular structure and the output is its prediction of Lipophilicity. A dictionary is used to map the symbols in the SMILES string to integers. We use the length of the longest SMILES in training dataset as the number of cells for the LSTM layer. Similar to text processing in the IMDB model, short SMILES inputs are padded with 0s to the left side. We use the root mean square error (RMSE) as the measurement of model accuracy. Our trained model achieves RMSE $= 0.2371$ in training dataset and RMSE $= 0.6278$ in test dataset, which are better than the traditional and convolutional methods used in [100].

Video Recognition for Human Behaviour A large scale VGG16+LSTM network is trained over the UCF101 dataset [101]. VGG16, a CNN for ImageNet, extracts features from individual frames of a video. Then, the sequence of frame features are analysed by LSTM layer for classification.

Test Metrics

We conduct experiments on several concrete test metrics, i.e., BC (for $\xi_t^{f,avg}$), SC (for $\Delta\xi_t^h$), and TC (for ξ_t^h). The configuration of thresholds are presented in Table 3.2. Although the proposed three test metrics can be applied to every internal vector of LSTM cell, like

f, i, o, c, h, the current settings represent better semantic meanings. The interpretation of the testing results is discussed in Section 3.7.7.

Input Mutation

For MNIST model, we add Gaussian noise to input image and round off the decimals around 0 and 1 to make the pixel value stay within the value range.

The input to IMDB model is a sequence of words, on which a random change may lead to an unrecognisable (and invalid) text paragraph. To avoid this, we take a set $\mathcal{M}$ of mutants from the EDA toolkit [102], which was originally designed to augment the training data for improvement on text classification tasks. This ensures the mutated text paragraphs are always valid. In our experiments, we consider four mutation operations, i.e., $\mathcal{M}$ includes (1) Synonym Replacement, (2) Random Insertion, (3) Random Swap, and (4) Random Deletion. The text meanings are reserved in all mutations. For Lipophilicity model, we take a set $\mathcal{M}$ of mutants which change the SMILES string without affecting the molecular structure it represents. The enumeration of possible SMILES for a molecule is implemented with the Python cheminformatics package RDkit [103]. Each input SMILES string is converted into its molfile format, based on which the atom order is changed randomly before converting back. There may be several SMILES strings representing the same molecular structure. The enumerated SMILES strings are the test cases.

For UCF101 model, we add Gaussian noise to the original video frames instead of the feature inputs to the LSTM layer.

Oracle Setting

We use one fixed oracle radius for each model across all experiments. For continuous inputs, like images and videos, we calculate the euclidean distance as the measurement of perturbation. For the discrete inputs, like text, We refer to the alpha parameter provided by the EDA toolkit, which approximately means the percent of words in the sentence that will be changed. That said, the r_{oracle} for each RNN are listed in Table 3.1. Note, we let r_{oracle} = None for the Lipophilicity model, suggesting that no constraint is imposed on the norm ball. Hence, the determination of adversarial example is completely based on the classification. This is because, as suggested before, the test cases are only generated from those SMILES strings with the same molecular structure.

Table 3.1: Summary of RNN models under testing

Test Model	No. of Classes	Test Acc.	Seq. of Interest	oracle
MNIST	9	98.7%	[4,24]	0.01
IMDB	2	86.2%	[400,500]	0.05
Lipophilicity	None	RMSE = 0.6278	[60,80]	None
UCF101	101	88.6%	[1,11]	0.1

3.7.2 Diversity of Test Cases

Table 3.2: Configuration of test metrics

Coverage Metrcis	Parameter Configuration
Neuron Coverage (NC)	Threshold = 0
K-multisection Neuron Coverage (KMNC)	$k = 10$
Neuron Boundary Coverage (NBC)	LB = -0.7, UB = 0.7
Strong Neuron Activation Coverage (SNAC)	UB = 0.7
Boundary Coverage (BC)	$\alpha_{max} = 0.8$
Step-wise Coverage (SC)	$\alpha_{sc} = 0.6$
Temporal Coverage (TC)	$w = 5$, $\lvert\Gamma\rvert = 3$

Test metrics can be seen as a proxy to exploit the input space, and intuitively more diversified test cases will explore more input space and thus are more likely to uncover different defects. Thus, we investigate if the achievement of high coverage will indeed lead to the testing of different LSTM functions (**RQ1**), and if our new metrics encourage the exploitation of more regions in the input space than existing metrics (**RQ2**).

Approximation of LSTM functional coverage (RQ1)

Table 3.3: Impact of seeds to coverage metrics

Test Model	Seeds Input & Test Cases	Categories of Seeds	Coverage Metrics						
			NC	KMNC	NBC	SNAC	BC	SC	TC
MNIST	100 / 5000	1	0.93	0.65	0.41	0.44	0.10	0.43	0.38
		10	**1.00**	**0.88**	**0.77**	**0.80**	**0.95**	**0.86**	**0.79**
IMDB	100 / 5000	1	1.00	0.29	0.01	0.01	0.23	0.45	0.24
		2	**1.00**	**0.37**	**0.01**	**0.01**	**0.81**	**0.54**	**0.64**
Lipophilicity	10 / 2000	1	0.81	0.31	0.05	0.06	0.00	0.00	0.04
		10	**1.00**	**0.86**	**0.68**	**0.66**	**0.95**	**0.95**	**0.90**
UCF101	100 / 5000	1	1.00	0.47	0.07	0.06	0.00	0.00	0.16
		10	**1.00**	**0.76**	**0.36**	**0.34**	**0.58**	**0.58**	**0.67**

Table 3.3 shows that LSTM model's functional coverage can be approximated by using TESTRNN metrics. This is based on the assumption that a data label (i.e., category) corresponds to a "functional feature" of the LSTM. We observe that, by only using one category of seeds input, it is hard to achieve high coverage rate for TESTRNN metrics, even when thousands of test cases are generated. In contrast, with seeds input from more categories, the generated test cases from targeted mutation can broadly explore the input space and more internal behaviours of RNNs. Thus, all rates of TESTRNN coverage metrics are significantly improved, given the test set. Table 3.3 also records the coverage of neuron level metrics, which are widely used in the CNNs/FNNs. These test metrics show less

sensitivity with respect to the diversity of functional features in the test suite, e.g., the NC coverage can already reach almost 100% by only using test cases of one label.

Answer to RQ1: The exploitation of internal behaviour by testRNN can approximate the testing of different LSTM functional features.

Comparison with Neuron Level Coverage (RQ2)

We implement the Neuron Coverage (NC) [104], k-multisection Neuron Coverage (KMNC), Neuron Boundary Coverage (NBC) and Strong Neuron Activation Coverage (SNAC) [55] on the testing layers of our LSTM models. We note that the concept "neuron" is ambiguous in RNNs, since the hidden output of RNNs' cells are vectors. Here, we consider covering each element of the hidden output h in the testing layer. Results are presented in Table 3.3 and 3.4.

Table 3.4: Complementarity of test metrics: comparison between neuron level test metrics and the proposed testRNN metrics in minimal test suite

Test Model	Target Metrics	Neuron Level Metrics				testRNN Metrics		
		NC	KMNC	NBC	SNAC	BC	SC	TC
MNIST	NC	**1.00**	0.61	0.39	0.44	0.10	0.00	0.10
	KMNC	1.00	**0.85**	0.67	0.72	0.10	0.29	0.44
	NBC	1.00	0.77	**0.73**	0.75	0.14	0.19	0.28
	SNAC	0.98	0.73	0.62	**0.75**	0.10	0.14	0.19
IMDB	NC	**1.00**	0.23	0.00	0.00	0.00	0.00	0.02
	KMNC	1.00	**0.39**	0.01	0.01	0.01	0.02	0.07
	NBC	0.48	0.13	**0.01**	0.01	0.00	0.00	0.01
	SNAC	0.47	0.10	0.01	**0.01**	0.00	0.00	0.00
Lipophilicity	NC	**1.00**	0.43	0.16	0.16	0.05	0.05	0.03
	KMNC	1.00	**0.92**	0.70	0.69	0.40	0.20	0.38
	NBC	1.00	0.84	**0.81**	0.78	0.50	0.20	0.22
	SNAC	1.00	0.77	0.62	**0.78**	0.40	0.20	0.13
UCF101	NC	**1.00**	0.48	0.26	0.36	0.10	0.10	0.15
	KMNC	1.00	**0.74**	0.60	0.66	0.18	0.10	0.20
	NBC	1.00	0.65	**0.75**	0.58	0.15	0.18	0.16
	SNAC	1.00	0.76	0.68	**0.84**	0.22	0.18	0.20

In the experiments, we find that NC can be trivially achieved. Shown in Table 3.3, NC is not suitable for exploring RNNs' internal functionality, since one category's seeds input is enough for the high coverage of neuron activation. Moreover, we find that other neuron level test metrics may be impossible to satisfy for IMDB test model. The low coverage rate of KMNC, NBC and SNAC indicates that the activation of neurons for IMDB model is concentrated in a small interval. In other words, the neuron level test metrics cannot be a good option to search for diverse test cases.

Table 3.4 shows the complementarity of neuron level test metrics and our proposed testRNN metrics. A set of complementary test metrics (and test cases) can enhance the diversity of the testing. In the experiments, we take **minimal test suite**, in which the removal of any test case may lead to the reduction of coverage rate. The consideration of the minimality of test suite enables a fair comparison since it reduces the overlaps as much as possible. The results confirm that a test suite which can achieve high coverage for neuron level test metrics is not necessary to get the high coverage for RNN test metrics. For example, in the MNIST LSTM model, test cases that achieve 100% NC can only cover less than 10% of the overall test conditions by testRNN metrics (with 10% BC, 0% SC and 10% TC). Similar patterns also happen to other models. That means the proposed test metrics provide the guide for the selection of additional test cases, which are complementary to those guided by the neuron level test metrics.

Moreover, we discover that there are many redundancy of test requirements, regarding to the relation between individual test metric in neuron level category. For example, if we derive a test suite which targets at increasing the coverage of KMNC, NBC and SNAC both get the high coverage results.

Answer to RQ2: The testRNN metrics exhibit a dramatic portion of LSTM internal behaviours that cannot be explored by existing metrics.

3.7.3 Detecting RNN Defects

Searching for Adversarial Samples (RQ3)

We collect the set of normal perturbed samples (N) and adversarial samples (A), respectively. First, normal perturbed samples are added to the test set to witness the increase of the coverage. When the coverage is difficult to improve, adversarial samples are considered. The update of whole process is illustrated in Fig. 3.5. The dashed vertical line distinguish the coverage update with normal perturbed samples from that with adversarial samples. It should be noted that the coverage update of some test metrics is stepped growth, due to the small amount of total test conditions which is shwon in Table 3.1.

Fig. 3.5 reveals that normal perturbed samples can only satisfy part of test conditions, while the rest are more sensitive to the adversarial samples. In all the plots, coverage of RNN test metrics can be further increased in consideration of adversarial samples. A more obvious example is, the TC coverage of IMDB model tend to saturate in the left side when only normal perturbed samples are utilized. In the right side, the coverage curve becomes steep, indicating the discovery of test cases capturing new internal behaviors.

In addition to the sensitivity of test metrics to adversarial samples, we show how to compare the robustness of models via coverage guided testing. We use TC as the termination condition to generate a test suite and calculate the adversarial rate. To achieve the high coverage of test metrics, we use genetic algorithm for test case generation, more details of which can be seen in Section 3.6.2. The other settings remain the same for the fair

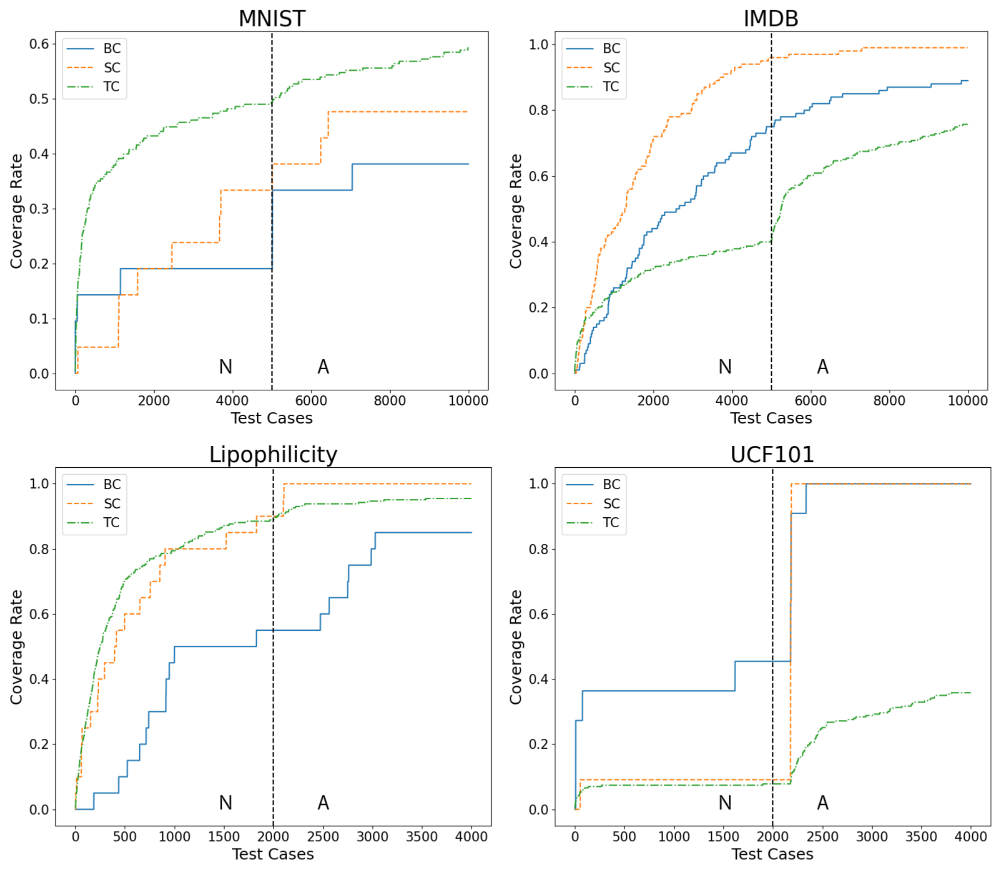


Figure 3.5: Update of coverage with normal perturbed samples ('N') and adversarial samples ('A')

comparison.

Table 3.5: Comparing the robustness of models via coverage guided testing

Test Dataset	Model No.	Test Cases	Adv. Samples Rate	Unique Adv. Samples	Coverage Metrics BC	SC	TC
MNIST	1	5958	**0.060**	**176**	0.48	0.81	0.90
	2	3570	0.075	184	0.57	0.86	0.90
IMDB	1	5841	**0.039**	138	0.94	0.93	0.75
	2	1575	0.047	**68**	0.62	0.72	0.75
Lipophilicity	1	2936	0.371	191	0.95	1.00	0.95
	2	6727	**0.010**	**44**	0.88	1.00	0.95
UCF101	1	6352	0.420	182	0.98	0.95	0.60
	2	6100	**0.250**	**90**	0.95	0.92	0.60

As shown in Table 3.5, adversarial samples rate and number of unique adversarial samples in the generated test suite are two important indicators for the robustness evaluation. The unique adversarial samples refer to the adversarial samples crafted from distinct seeds input. For a set of trained models, we pursue the model, the test suite of which contains less amount of adversarial samples and unique adversarial samples. For example, we pick up model 2 for ipophilicity prediction, since the values of two indicators are way smaller than that of model 1. We comment that with large enough amount of test cases, coverage-guide testing approach provides a new way for the measure and selection of more robust classifier. This is compatible with the results in [5] that a poorly trained neural network exposes more adversarial samples subject to well-defined coverage guided testing.

Answer to RQ3: By exploiting the model's internal behaviours, testRNN is able to capture the LSTM adversarial samples.

Detecting Backdoor input in RNNs (RQ4)

We investigate the possibility of applying coverage-guided testing to the detection of backdoor input in neural networks. We try to exploit if there is any difference between clean input and backdoor input which can be captured by our proposed test metrics. Examples of backdoor input are illustrated in Fig. 3.6. We train two handwritten digits recognition models, one of which is benign classifier and the other one is the malicious classifier subject to the backdoor attack in [79]. Table 3.6 shows that, both benign and malicious classifiers keep good prediction performance in clean test set. For the backdoor test set, benign classifier keep the normal accuracy, while the malicious classifier predicts inputs with the backdoor trigger as the attacked label successfully.

Table 3.6: Sensitivity of test metrics to backdoor samples in MNIST dataset

Model	Test Acc. (C / B)	Data	Class 0			Class 6			Class 9		
			BC	SC	TC	BC	SC	TC	BC	SC	TC
Benign	99.1% / 9.5%	T	0.39	0.25	0.16	0.29	0.18	0.30	0.32	0.29	0.22
		T + C	0.39	0.25	0.17	0.29	0.18	0.30	0.32	0.29	0.22
		T + B	0.39	0.25	0.17	0.29	0.18	0.30	0.32	0.29	0.22
Malicious	98.7% / 100%	T	0.39	0.18	0.30	0.25	0.18	0.60	0.07	0.18	0.27
		T + C	0.39	0.18	0.30	0.25	0.18	0.60	0.07	0.18	0.27
		T + B	0.39	**0.25**	**0.33**	0.25	**0.21**	**0.63**	0.07	**0.21**	**0.29**

We conduct sensitivity analysis by computing the coverage of the proposed test metrics in training data (T), clean test data (C), and backdoor test data (B) for each classifier. In the first row of Table 3.6, we calculate the coverage of the training data from same class. On the basis of this, we add clean test data or backdoor test data for evaluation. If the coverage rate is further increased in second and third row, the new internal patterns are discovered. The experimental results describe that backdoor input activate same internal behavior with clean input for a benign classifier. In contrast to this, the backdoor input to malicious classifier will induce different internal activation, which can be seen from the apparent increase of coverage in T+B. Although the backdoor input is very similar to the clean input with a small region of pixels changed (Fig. 3.6), the internal activation in the malicious model can still be revealed by the coverage change of the proposed TESTRNN metrics.

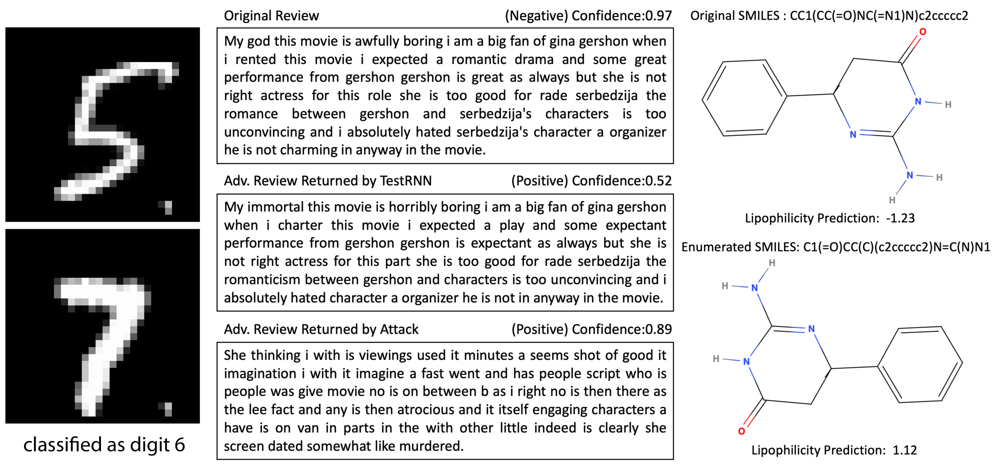
We remark that the above experiment only confirms that test metrics are sensitive to backdoor samples when testing an attacked model. More accurate detection of backdoor in RNNs needs more precise refinement of test metrics, e.g. adding the backdoor knowledge to the metrics design on top of the structure information. Nevertheless, the goal of coverage guided testing is still diversifying the test suite so that defects like backdoor samples are more likely to be detected.

Answer to RQ4: The TESTRNN metrics can identify the difference between the backdoor input and the normal input (to malicious models).

3.7.4 Effectiveness of Test Case Generation (RQ5)

We show the effectiveness of our test case generation from the following aspects: (1) it is non-trivial to achieve high coverage rate, and (2) there is a significant percentage of adversarial samples in the generated test suite. For (1), we show that the targeted mutation (i.e., random mutation enhanced by genetic algorithm) is needed to boost the coverage rate. Three test case generation methods are considered: (Seeds) sampling 200 seeds input from training dataset, (Ran.) generating test cases from seeds by using random mutation, and (Targ.) generating test cases from seeds by using targeted mutation. Fig. 3.6 demonstrates detected adversarial samples for IMDB and Lipophilicity models, and we omit other models for brevity. All experimental results are based on 5 runs with different random seeds. The results are averaged and summarised in Table 3.7. For each test case generation method and LSTM model, we also report the number of adversarial samples, unique adversarial samples in the test suite and their average perturbation. This experiment considers all four models.

Table 3.7 shows that, the coverage rates and the number of adversarial samples for Ran. are significantly higher than those of Seeds, that is, Ran. is effective in finding the adversarial samples around the original seeds. Furthermore, if we use Targ., both the coverage rates and the number of adversarial samples are further increased. The above observations confirm the following two points: (1) our test metrics come with a strong bug finding ability; and (2) higher coverage rates indicate more comprehensive test. We remark that, the TC rates for UCF101 model are relatively low and harder to improve, because the mutations are made on the image frames (i.e., before CNN layers) instead of directly on the LSTM input. This shows that the adversarial samples for CNNs are orthogonal to those of LSTMs, another evidence showing that test metrics for CNNs cannot be directly applied to RNNs.


classified as digit 6
Original SMILES : CC1(CC(=O)NC(=N1)N)c2ccccc2
Lipophilicity Prediction: -1.23
Enumerated SMILES: C1(=O)CC(C)(c2ccccc2)N=C(N)N1
Lipophilicity Prediction: 1.12

Original Review (Negative) Confidence:0.97

My god this movie is awfully boring i am a big fan of gina gershon when i rented this movie i expected a romantic drama and some great performance from gershon gershon is great as always but she is not right actress for this role she is too good for rade serbedzija the romance between gershon and serbedzija's characters is too unconvincing and i absolutely hated serbedzija's character a organizer he is not charming in anyway in the movie.

Adv. Review Returned by TestRNN (Positive) Confidence:0.52

My immortal this movie is horribly boring i am a big fan of gina gershon when i charter this movie i expected a play and some expectant performance from gershon gershon is expectant as always but she is not right actress for this part she is too good for rade serbedzija the romanticism between gershon and characters is too unconvincing and i absolutely hated character a organizer he is not in anyway in the movie.

Adv. Review Returned by Attack (Positive) Confidence:0.89

She thinking i with is viewings used it minutes a seems shot of good it imagination i with it imagine a fast went and has people script who is people was give movie no is on between b as i right no is then there as the lee fact and any is then atrocious and it itself engaging characters a have is on van in parts in the with other little indeed is clearly she screen dated somewhat like murdered.

Figure 3.6: Backdoor samples for MNIST model (left). Adversarial samples for IMDB (middle) and Lipophilicity (right) models.

Answer to RQ5: The test case generation algorithm is effective in improving both the coverage rate and the adversary rate. In particular, the targeted mutation method can be utilised to find more corner samples.

Table 3.7: Experiments for Test Case Generation Methods

Test Model	Test Gen. Method	Test Cases	No. of Adv. samples	Avg. Perturb.	Unique Adv. Samples	Coverage Metrics		
						BC	SC	TC
MNIST	Seeds	200	-	-	-	0.43	0.14	0.34
	Ran.	10000	226	1.180	18	0.57	0.52	0.66
	Targ.	10000	**244**	**1.497**	**32**	**1.00**	**1.00**	**0.79**
IMDB	Seeds	200	-	-	-	0.11	0.05	0.24
	Ran.	10000	308	0.136	88	0.84	0.40	0.77
	Targ.	10000	**367**	**0.103**	**97**	**1.00**	**0.58**	**0.82**
Lipophilicity	Seeds	200	-	-	-	0.65	0.55	0.48
	Ran.	2000	812	-	190	0.95	1.00	0.91
	Targ.	2000	**834**	-	**194**	**1.00**	**1.00**	**0.95**
UCF101	Seeds	200	-	-	-	0.52	0.53	0.11
	Ran.	10000	3613	1.031	112	0.82	0.90	0.31
	Targ.	10000	**4201**	**1.251**	**156**	**1.00**	**1.00**	**0.66**

3.7.5 Comparison with Attack-based Defect Detection (RQ6)

We compare testRNN with state-of-the-art RNN adversarial attack [99, 105], which detects robustness defects. These attack algorithms utilise the model's gradient over input sequence to iteratively change some parts of the input that contribute the most to the model's prediction. Their methods can successfully find adversarial samples. However, these attack methods have two main drawbacks, when compared with our testing method.

First, attack methods search for adversarial samples by adding perturbations in the gradient direction. This easily leads to the situation where the generated adversarial samples are concentrated in a "buggy" area of the input space, as shown in Table 3.9 and Fig. 3.7. We first collect the same amount of adversarial samples in MNIST model returned by attack methods and testRNN, respectively. Then, we calculate the angular-based diversity of each set (Table 3.9) and apply the Principal Component Analysis (PCA), a well known dimensionality reduction technique, to project the high dimensional adversarial images onto two dimensional space for better visualisation (Fig. 3.7). We can see from the resulting diversity measurement and visualisation that, compared to attack methods, our testing method exercises different behaviors of RNN and generates a diverse set of test cases, intensively covering the input region around the seed input. *This ability will be helpful in exposing more types of defects of the RNN* (not merely in the gradient direction).

Moreover, RNNs are widely applied to the nature language processing, in which the inputs to an RNN, i.e., words, are discretely distributed. Attack methods aggressively replace important words in the text and produce an adversarial sequence. In this process, it is hard to consider both the gradient and the whole text's semantic meaning. That is, the modified text may easily become human-unreadable and impossible to occur in real world. On the other hand, our testing method is able to reduce such problems by taking the mutants from off-the-shelf tools such as the EDA toolkit. Fig. 3.6 presents adversarial movie reviews returned by attack method and testRNN, respectively. It is easy to see that the adversarial review returned by the gradient attack is hard to comprehend while the one from testRNN is much easier.

Answer to RQ6: The testRNN is able to generate a set of diverse and natural test cases, so as to expose more types of defects.

3.7.6 Comparison with State-of-the-Art testing methods (RQ7)

We compare testRNN with DeepStellar, a state-of-the-art testing tool dedicated for RNNs. As discussed in Section 3.5, two different test metrics are integrated in DeepStellar, i.e. state coverage and transition coverage, which are corresponding to boundary coverage and stepwise coverage in testRNN. Apart from these, testRNN have temporal coverage for the internal sequential processing behaviour of RNNs. We start from 100 seeds drawn from training set and generate 100000 test cases by DeepStellar and testRNN, respectively. The test suites are evaluated for the coverage rate and number of adversarial samples. We compute basic state coverage (BS), basic transition coverage (BT), and weighted transition coverage (WT) in DeepStellar guided by different generation strategy, S-Guide and T-Guide. The testing results for both are recorded in Table 3.8. First, we can see that test metrics in DeepStellar already have high coverage rates upon seeds input, as opposed to our metrics which display relatively smaller coverage rates upon the same seeds. That means that our metrics are better for exploiting the input space around seeds. Second, DeepStellar adopts the fuzzing strategy with the guidance of different test metrics, which is effective to boost the coverage. However, in this experiment for small-scale model trained on MNIST, 100000 test cases are still not enough for 100% coverage of test requirements in DeepStellar. It seems that some of their defined test requirements may be infeasible to satisfy. On the contrary, testRNN can achieve a relatively high coverage results with random mutation and the coverage rates of all the metrics can be significantly boost to achieve 100% by genetic algorithm based mutation method. The number of adversarial samples in the test suite reflects that testRNN is superior to DeepStellar in terms of exploiting diverse internal behaviors and bugs finding ability.

Table 3.8: Comparison between DeepStellar and testRNN using MNIST: 100000 test cases are generated from 100 seeds

DeepStellar				TestRNN			
Test Metrics	Seeds	S-Guid.	T-Guid.	Test Metrics	Seeds	Ran.	Targ.
BS	0.45	0.80	0.82	BC	0.14	0.57	1.00
BT	0.11	0.32	0.63	SC	0.38	0.67	1.00
WT	0.76	0.90	0.95	TC	0.24	0.70	1.00
Adv. Samples	-	1588	1661	Adv. Samples	-	1778	1830

Table 3.9: Angular-based diversity (a greater value represents a better diversity) and average perturbation (smaller is better) of adversarial samples

Seed	Angular-based Diversity			Avg. Perturb.		
	testRNN	DeepStellar	Attack	testRNN	DeepStellar	Attack
1	-0.277	-0.468	-0.598	0.006	0.012	0.014
2	-0.289	-0.438	-0.556	0.006	0.010	0.013

To understand the relative merits of the defects returned by DeepStellar, testRNN, and Gradient-based Attack, respectively, we compute the angular-based diversity and average perturbation of each set (Table 3.9), and visualise them with PCA projection (Fig. 3.7). We can see that the adversarial samples from DeepStellar are sparsely distributed and most of them are more distant to the seed input. testRNN explores space that is close to the seed input. This aligns better to the goal of adversarial testing, which is to find more bugs around the seed with as small perturbations as possible (the bugs are more realistic/natural, thus more likely to exist in real world).

In addition to the comparison of testing results, we are also interested in the complementarity of test metrics in testRNN and DeepStellar. We derive the minimal test suites by different test case generation methods. Then, the test suite generated by testRNN is evaluated for the coverage of metrics in DeepStellar, and vice versa. Results in Table 3.10 suggests that test suite generated by testRNN can easily achieve high coverage rate for the

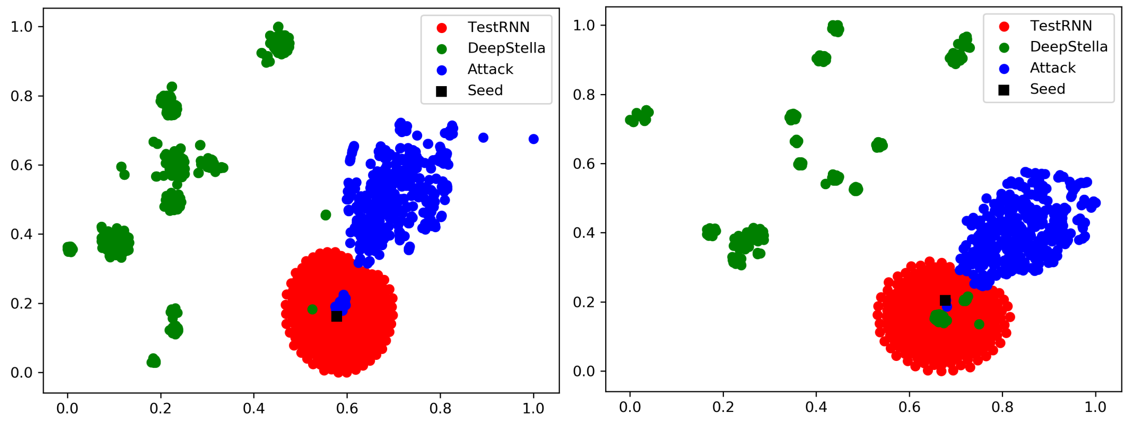


Figure 3.7: Visualisation of adversarial samples generated by TESTRNN, DeepStellar, and Gradient-based Attack, respectively, in MNIST model. The visualisation is conducted by projecting high-dimensional images onto a two-dimensional space. Each figure corresponds to a seed input in the dataset.

Table 3.10: Complementarity of test metrics in DeepStellar and TESTRNN

Tool	Test Gen. Method	Target Metrics	Coverage Metrics BS	BT	WT	BC	SC	TC
TestRNN	Ran.	-	0.78	0.63	0.94	0.57	0.67	0.70
	Targ.	BC,SC,TC	0.78	0.64	0.95	1.00	1.00	1.00
DeepStellar	S-Guide.	BS	0.80	0.32	0.90	0.05	0.10	0.12
	T-Guide.	BT,WT	0.82	0.63	0.95	0.10	0.24	0.20

metrics in DeepStellar. We find that, test suite produced by DeepStellar cannot get high coverage rate on our metrics. This confirms our discussion of the relation between coverage metrics in Section 3.5.

Answer to RQ7: The TESTRNN test generation can achieve high coverage of the test metrics in DeepStellar, but not vice versa.

3.7.7 Exhibition of Internal Working Mechanism (RQ8)

In this section, we show that the working mechanism of LSTM networks can be understood via the test coverage results generated from TESTRNN. We conduct experiments to visualise the learning process of LSTM layer via TESTRNN results.

Coverage times denote the number of times a test condition is satisfied by running the test suite. Intuitively, coverage times represent the level of difficulty of asserting an input feature. Fig. 3.8 reports the coverage times for each input feature. We note that, in BC and SC, each input feature x_t corresponds to a test condition on $\xi_t^{s,a}$, as in MNIST it is defined with respect to a row of pixels on the input image. In sentiment analysis model, the input feature refers to a word in movie reviews.

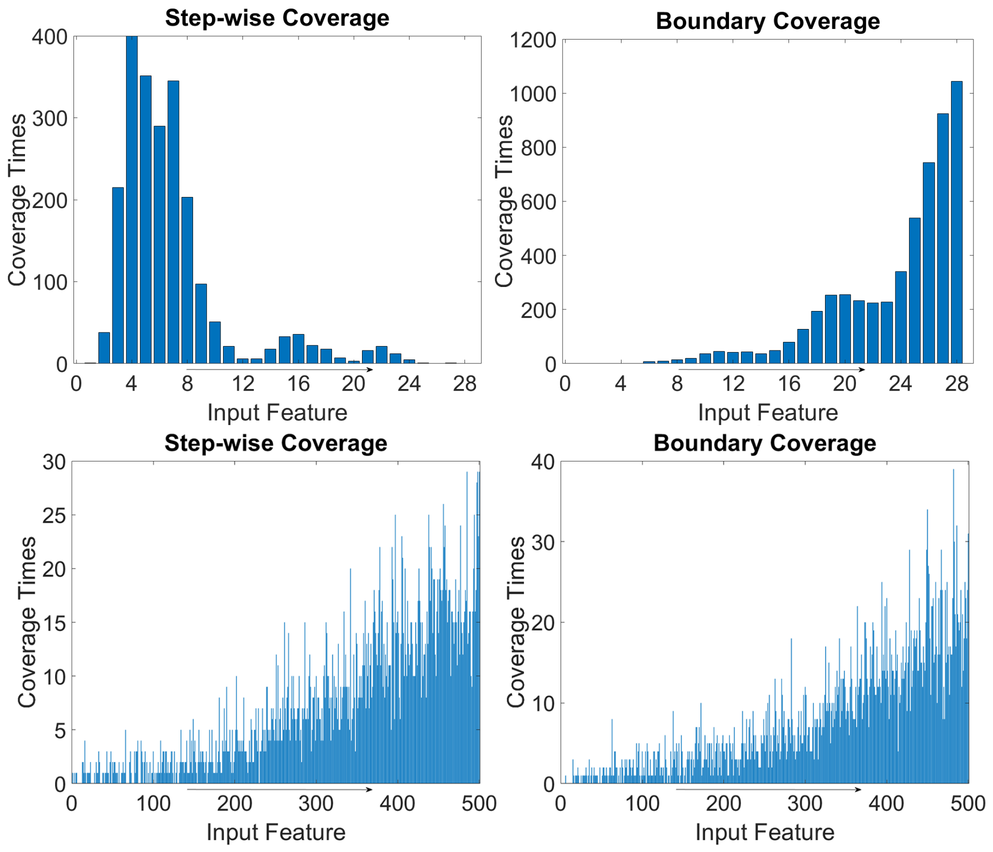


Figure 3.8: 2000 test cases are used to demonstrate the coverage times of 28 features in an LSTM layer of MNIST model (first line) and 500 input features in LSTM layer of IMDB Sentiment Analysis model (second line).

As discussed in Section 3.4, SC is to assert if an input feature is significant to the model prediction. Then an important input feature will cause great changes of hidden memory h_t and satisfy the test condition of SC. BC monitors the forget gate values at each time step. The satisfaction of BC means the LSTM will not drop out the information stored in memory.

If we combine SC and BC plots, the whole working process of LSTM layer inside the MNIST model becomes transparent. The sequential input of an image starts from the top row and finishes at the bottom row. At the beginning, the top rows of MNIST images are blank and do not contribute to the model prediction. These less-important information is gradually thrown away from the memory. When the input rows containing digits are fed to the LSTM cells, the model will start learning and the short term memory, represented by the outputs h_t, start to have strong reactions. When approaching the end of the input, which corresponds to the bottom of the digit images, LSTM has already been confident about the final classification and therefore becomes lazy to update the memory. Overall, we can see that, MNIST digits recognition is not a complicated task for the LSTM model and usually the top half of the images are sufficient for the classification.

For the IMDB model, the final classification is influenced by every input feature. To make sure that input features between 450-500 contain real words instead of padded 0s, we take 2,000 reviews whose length are greater than 50. We observe from the second line in Fig. 3.8 that the coverage times gradually increase, it might be the nature of test cases – most test cases contain text of length much less than 500. We therefore focus on the last 50 input features. We see that, both BC and SC test conditions in the IMDB model are randomly activated, a phenomenon that is completely different from that of MNIST results. This can be explained as that the IMDB model does not have a fixed working pattern like the MNIST model. Sensitive words in a review may appear in any place of the text.

Answer to RQ8: The generated test suite can be used to understand the data processing mechanism of LSTM models. This is a step towards interpretable RNNs.

3.7.8 Threats to Validity

First, we fix the thresholds or symbols of test metrics for all the experiments. If we decrease the values of threshold (or reduce the symbols to represent sub-ranges), the test conditions can be easier to satisfy, and fewer test cases are generated. Conversely, if we tighten the thresholds, more test cases are needed to cover the test conditions. The input space are more thoroughly explored.

Second, we only choose part of input sequence to test, details of which is shown in Table 3.1. If we use testRNN to test the entire input sequence, some test conditions may be harder to meet. The choices of partial input sequences in our experiment are as follows. For MNIST dataset, the hand-written digits are usually concentrated on 4th to 24th rows out of 28 rows. The rest of the images are blank. For IMDB and Lipophilicity dataset, the input to RNNs are usually padded with 0s. And the input 0s only induce very small activation, which can be seen in Fig. 3.8.

We define unique mutation functions for different models to ensure the generated test input are always valid. Since we set thresholds of test metrics with reference to the training data, it is non-trivial to validate the test input. Mutation function needs to keep the semantics meanings of seeds input. For example, in the experiment of testing IMDB model, we mutate the text paragraph instead of the input to LSTM layer.

Some minor threats include the settings of oracle and random seeds. We also fix the configurations for these parameters to make all the experiments consistent. The oracle radius can affect the adversarial samples rate and the average perturbations in the test suite. If we set up a smaller oracle radius, the number of perturbed input recognized as adversarial samples and the average perturbations are both decreased. The random seeds are utilized to control the reproducibility of the experiments. In most experiments, we do several test with different seeds input and get the average results so that the accidental errors can be avoided.

Chapter 4

Test DL Robustness through Hierarchical Distribution-Awareness

4.1 Introduction

Deep Learning (DL) is being explored to provide transformational capabilities to many industrial sectors including automotive, healthcare and finance. The reality that DL is not as dependable as required now becomes a major impediment. For instance, key industrial foresight reviews identified that the biggest obstacle to gaining benefits of DL is its dependability [106]. There is an urgent need to develop methods to enable the dependable use of DL, for which great efforts have been made in recent years in the field of DL Verification and Validation (V&V) [49, 107].

As recently noticed by the software engineering community, emerging studies on systematically evaluating Adversarial Example (AE) detected by aforementioned state-of-the-arts have two major drawbacks: (i) they do not take the *input data distribution* into consideration, therefore it is hard to judge whether the identified AEs are meaningful to the DL application [29, 30]; (ii) most detected AEs are of *poor perception quality* that are too unnatural/unrealistic [31] to be seen in real-life operations. That said, not all AEs are equal nor can be eliminated given limited resources. A wise strategy is to detect those AEs that are both being "distribution-aware" and with natural/realistic pixel-level perturbations, which motivates this work.

Prior to this work, a few decent attempts at distribution-aware testing for DL have been made. Broadly speaking, the field has developed two types of approaches: *Out-Of-Distribution (OOD) detector* based [30, 108] and *feature-only* based [109, 110]. The former can only detect anomalies/outliers, rather than being "fully-aware" of the distribution. While the latter is indeed generating new test cases according to the learnt distribution (in a latent space), it ignores the pixel-level information due to the compression nature of generative models used [111]. To this end, our approach is advancing in this direction with the following novelties and contributions:

a) We provide a "divide and conquer" solution—Hierarchical Distribution-Aware (HDA) testing—by decomposing the input distribution into two levels (named as *global* and *local*) capturing how the feature-wise and pixel-wise information are distributed, respectively. At the global level, isolated problems of estimating the feature distribution and selecting best test seeds can be solved by dedicated techniques. At the local level where features are fixed, the clear objective is to precisely generate test cases considering perceptual quality. Our extensive experiments show that such *hierarchical* consideration is more effective to detect high-quality AEs than state-of-the-art that either disregards any data distribution or only considers a single (non-hierarchical) distribution. Consequently, we also show the DL model under testing exhibits higher robustness after "fixing" the high-quality AEs detected.

b) At the global level, we propose novel methods to select test seeds based on the *approximated feature distribution* of the training data and *predictive robustness indicators*, so that the norm balls of the selected seeds are both from the high-density area of the distribution and relatively unrobust (thus more cost-effective to detect AEs in later stages). Notably, state-of-the-art DL testing methods normally select test seeds *randomly* from the training dataset without any principled rules. Thus, from a software engineering perspective, our test seed selection is more practically useful in the given application context.

c) Given a carefully selected test seed, we propose a novel two-step Genetic Algorithm (GA) to generate test cases locally (i.e. within a norm ball) to control the *perceptual quality* of detected AEs. At this local level, the perceptual quality distribution of data-points inside a norm ball requires pixel-level information that cannot be sufficiently obtained from the training data alone. Thus, we innovatively use common perceptual metrics that quantify image quality as an approximation of such local distribution. Our experiments confirm that the proposed GA is not only effective after being integrated into HDA (as a holistic testing framework), but also outperforms other pixel level AE detectors in terms of perception quality when applied separately.

d) We investigate black-box (to the DL model under testing) methods for the main tasks at both levels. Thus, to the best of our knowledge, our HDA approach provides an *end-to-end, black-box* solution, which is the first of its kind and more versatile in software engineering practice.

e) A publicly accessible tool of our HDA testing framework with all source code, datasets, DL models and experimental results.

4.2 Preliminaries and Related Work

In this section, we first introduce preliminaries and related work on DL robustness, together with formal definitions of concepts adopted in our HDA approach. Then existing works on distribution-aware testing are discussed. Since our HDA testing also considers the naturalness of detected AEs, some common perception quality metrics are introduced. In summary, we present Fig. 4.1 to show the stark contrast of our proposed HDA testing (the green route) to other related works (the red and amber routes).

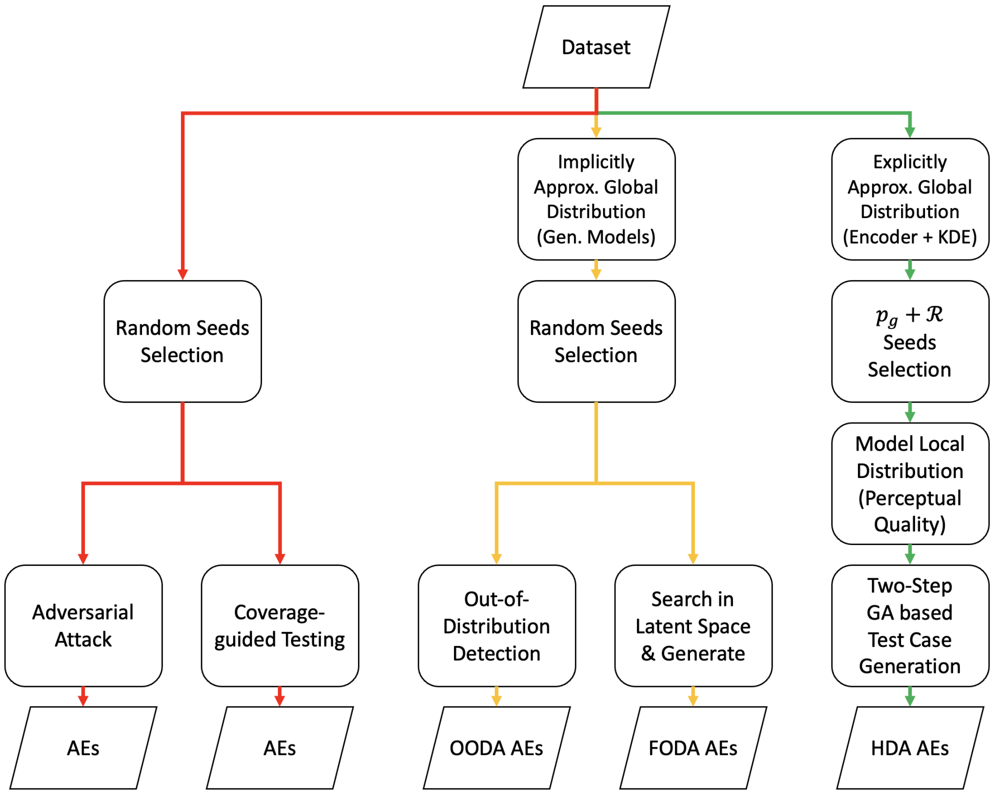

Dataset
Implicitly Approx. Global Distribution (Gen. Models)
Explicitly Approx. Global Distribution (Encoder + KDE)
Random Seeds Selection
Random Seeds Selection
$p_g + \mathcal{R}$ Seeds Selection
Model Local Distribution (Perceptual Quality)
Adversarial Attack
Coverage-guided Testing
Out-of-Distribution Detection
Search in Latent Space & Generate
Two-Step GA based Test Case Generation
AEs
AEs
OODA AEs
FODA AEs
HDA AEs

Figure 4.1: Comparison between our proposed Hierarchical Distribution-Aware (HDA) testing and related works.

4.2.1 DL Robustness and Adversarial Examples

We denote the prediction output of DL model as the vector $f(x)$ with size equal to the total number of labels. The predicated label $\hat{f}(x) = \text{argmax}_i\, f_i(x)$ where $f_i(x)$ is the i^{th} attribute of vector $f(x)$.

DL robustness requires that the decision of the DL model $\hat{f}(x)$ is invariant against small perturbations on input x. That is, all inputs in an input region η have the same prediction label, where η is usually a small norm ball (defined with some L_p-norm distance[1]) around an input x. If an input x' inside η is predicted differently to x by the DL model, then x' is called an *Adversarial Example* (AE).

DL robustness V&V can be based on formal methods [112, 113] or statistical approaches [114, 115], and normally aims at detecting AEs. In general, we may classify two types of methods (the two branches in the red route of Fig. 4.1) depends on how the test cases are generated: (i) Adversarial attack based methods are normally optimised for the DL prediction loss to find AEs, which include white-box attack methods like FGSM [1] and PGD [24], as well as black-box attacks [116, 117] using GA with gradient-free optimisation.

[1]$p = 0, 1, 2$ and ∞. L_∞ norm is more commonly used.

(ii) Coverage-guided testing are optimised for certain coverage metrics on the DL model's internal structure, which is inspired by the coverage testing for traditional software. Several popular test metrics, like neuron coverage [118, 55], modified condition/decision coverage [119] for CNNs and temporal coverage [120, 4] for RNNs are proposed. While it is argued that coverage metrics are not strongly correlated with DL robustness [121, 31], they are seen as providing insights into the internal behaviours of DL models and hence may guide test selection to find more diverse AEs [120].

Without loss of generality, we reuse the formal definition of DL robustness in [115, 122] in this work:

Definition 1 (Local Robustness)**.** *The local robustness of the DL model $f(x)$, w.r.t. a local region η and a target label y, is:*

$$\mathcal{R}_l(\eta, y) := \int_{x \in \eta} I(x) p_l(x \mid x \in \eta) \, \mathrm{d}x \tag{4.1}$$

where $p_l(x \mid x \in \eta)$ is the local distribution of region η which is precisely the "input model" used by both [115, 122]. $I(x)$ is an indicator function, and $I(x) = 1$ when $\hat{f}(x) = y$, $I(x) = 0$ otherwise.

To detect as many AEs as possible, normally the first question is—which local region shall we search for those AEs? I.e. how to select test seeds? To be cost-effective, we want to explore unrobust regions, rather than regions where AEs are relatively rare. This requires the local robustness of a region to be known *a priori*, which may imply a paradox (cf. Remark 3 later). In this regard, we can only *predict* the local robustness of some regions before doing the actual testing in those regions. We define:

Definition 2 (Local Robustness Indicator)**.** ***Auxiliary information*** *that strongly* ***correlated*** *with $\mathcal{R}_l(\eta, y)$ (thus can be leveraged in its prediction) is named as a local robustness indicator.*

We later seek for such indicators (and empirically show their correlation with the local robustness), which forms one of the two key factors considered in selecting test seeds in our method.

Given a test seed, we search for its AEs in a local region η that with different labels. This involves the question on what size of η should be, for which we later utilise the property of:

Remark 1 (r-separation)**.** *For real-world image datasets, any data-points with different ground truth labels are at least distance $2r$ apart in the input (pixel) space, with r being estimated case by case depends on the dataset.*

The r-separation property was first observed by [123]: intuitively it says, there is a minimum distance between two real-world objects of different labels.

Finally, not all AEs are equal in terms of the "strength of being adversarial" (stronger AEs may lead to greater robustness improvement in, e.g., adversarial training [124]), for which we define:

Definition 3 (Prediction Loss)**.** *Given a test seed x with label y, the prediction loss of an input x' to the test seed is defined as:*

$$\mathcal{J}(f(x'), y) = \max_{i \neq y}(f_i(x') - f_y(x')) \tag{4.2}$$

where $f_i(x')$ returns the probability of label i after input x' being processed by the DL model f.

Note, $\mathcal{J} \geq 0$ implies $\text{argmax}_i\, f_i(x) \neq y$ and thus x' is an AE of x.

Next, to measure the DL models' *overall* robustness across the whole input domain, we introduce a notion of global robustness. Being different to some existing definitions where local robustness are treated equally [125, 22], ours is essentially a "weighted sum" of the local robustness of local regions where each weight is the probability of the associated region on the input data distribution. Defining global robustness in such a "distribution-aware" manner aligns with our motivation—as revealed later by empirically estimated global robustness, our HDA appears to be more effective in supporting the growth of the overall robustness after "fixing" those distribution-aware AEs.

Definition 4 (Global Robustness)**.** *The global robustness of the DL model $f(x)$ is defined as:*

$$\mathcal{R}_g := \sum_{\eta \in \mathcal{X}} p_g(x \in \eta)\mathcal{R}_l(\eta, y) \tag{4.3}$$

where $p_g(x \mid x \in \eta)$ is the global distribution of region η (i.e., a pooled probability of all inputs in the region η_z) and $\mathcal{R}_l(\eta, y)$ is the local robustness of region η to the label y.

The estimation of $\mathcal{R}_g$, unfortunately, is very expensive that requires to compute the local robustness $\mathcal{R}_l$ of a large number of regions. Thus, from a practical standpoint, we adopt an empirical definition of the global robustness in our later experiments, which has been commonly used for DL robustness evaluation in the adversarial training [126, 24, 124, 127].

Definition 5 (Empirical Global Robustness)**.** *Given a DL model f and a validation dataset D_v, we define the empirical global robustness as $\hat{\mathcal{R}_g} : (f, D_v, T) \to [0, 1]$ where T denotes a given type of AE detection method and $\hat{\mathcal{R}_g}$ is the weighted accuracy on AEs obtained by conducting T on $\langle f, D_v \rangle$.*

To be "distribution-aware", the synthesis of D_v should conform to the global distribution while locally AEs are searched by T according to the local distribution. Consequently, the set of AEs may represent the input distribution.

4.2.2 Distribution-Aware Testing for DL

There are increasing amount of DL testing works developed towards being distribution-aware (as summarised in the amber route of Fig. 4.1). Deep generative models, such as Variational Auto-Encoders (VAE) and Generative Adversarial Networks (GAN), are applied to approximate the training data distribution, since the inputs (like images) to Deep Neural Network (DNN) are usually in a high dimensional space. Previous works heavily rely on OOD detection [30, 108] or synthesising new test cases directly from latent spaces [128, 110, 109, 129]. The former not really considers the whole distribution, rather flags outliers, thus a more pertinent name of it should be *out-of-distribution-aware* (OODA) testing. While for both types of methods, another problem arises that the distribution encoded by generative models only contain the *feature-wise* information and filter out the *pixel-wise* perturbations [111]. Consequently, directly searching and generating test cases from the latent space of generative models may *only perturb features*, thus called *Feature-Only Distribution-Aware* (FODA) in this paper (while also named as *semantic* AEs in some literature [130, 131]). Our approach, the green route in Fig. 4.1, differs from aforementioned works by considering both the global (feature level) distribution in latent spaces and the local (pixel level) perceptual quality distribution in the input space.

4.2.3 Perception Quality of Images

Locally, data-points (around the selected seed) sharing the same feature information may exhibit differently in terms of naturalness. To capture such distribution, some *perceptual quality* metric can be utilised to compare the perceptual difference between the original images and perturbed images. Some *common* metrics for perceptual quality include:

- Mean Square Error (MSE) between the original image and perturbed image.
- Peak Signal-to-Noise Ratio (PSNR) [132] defined as $20 * log_{10}\frac{MAX}{\sqrt{MSE}}$, where MAX is the maximum possible pixel value of the image.
- Structural Similarity Index Measure (SSIM) [133] that considers image degradation as perceived change in structural information.
- Fréchet Inception Distance (FID) [134] that compares the distribution between a set of original images and a set of perturbed images by squared Wasserstein metric.

Notably, all these metrics are current standards for assessing the quality of images, as widely used in the experiments of aforementioned related works.

4.3 The Proposed Method

We first present an overview of our HDA testing, cf. the green route in Fig. 4.1, and then dive into details of how we implement each stage by referring to an illustrative example in Fig. 4.2.

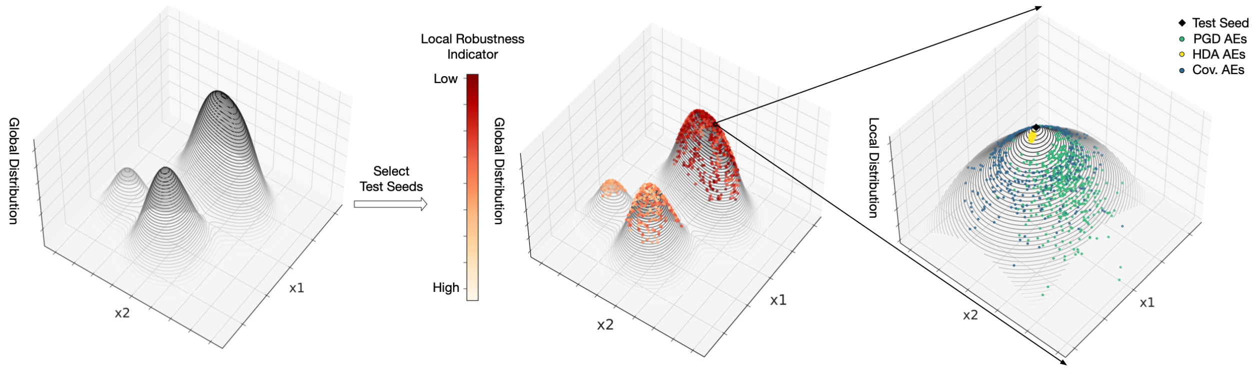


Figure 4.2: An example of Hierarchical Distribution Aware Testing

4.3.1 Overview of HDA Testing

The core of HDA testing is the hierarchical structure of two distributions. We formally define the following two levels of distributions:

Definition 6 (Global Distribution). *The global distribution captures how **feature** level information is distributed in some (low-dimensional) latent space after data compression.*

Definition 7 (Local Distribution). *Given a data-point sampled from the latent space, we consider its norm ball in the input **pixel** space. The local distribution is a conditional distribution capturing the perceptual quality of all data-points within the norm ball.*

Latent space is the representation of compressed data, in which data points with similar features are closer to each other. DNNs, e.g. encoder of VAEs, map any data points in the high dimensional input space to the low dimensional latent space. It infers that input space can be divided into distinct regions, and each region corresponds to a data point in latent space. By fitting a global distribution in the latent space, we actually model the distribution of distinct regions over the input space. The local distribution is defined as a conditional distribution within each region, sharing the same features. Thus, we propose the following remark.

Remark 2 (Decompose one distribution into two levels). *Given the definitions of global and local distributions, denoted as p_g and p_l respectively, we may decompose a single distribution over the entire input domain $\mathcal{X}$ as:*

$$p(x) = \int p_l(x|x \in \eta_z) p_g(x \in \eta_z) \, \mathrm{d}z \tag{4.4}$$

where variable z represents a set of features while η_z represents a region in the input space that "maps" to the z point in the latent space.

Intuitively, compared to modelling a single distribution, our hierarchical structure of distributions is superior in that the global distribution guides for which regions of the input space to test, while the local distribution can be leveraged to precisely control the perceptual quality of test cases. The green route in Fig. 4.1 shows our HDA testing process, which appears as three stages:

Stage 1: Explicitly Approximate the Global Distribution We first extract *feature-level* information from the given dataset by using data compression techniques—the encoder of VAEs in our case, and then explicitly approximate the global distribution in the latent-feature space, using Kernel Density Estimator (KDE).

Stage 2: Select Test Seeds Based on the Global Distribution and Local Robustness Indicators Given the limited testing budget, we want to test in those local input regions that are both more error-prone and representative of the input distribution. Thus, when selecting test seeds, we consider two factors—the local robustness indicators (cf. Definition 2) and the global distribution. For the former, we propose several auxiliary information with empirical studies showing their correlation with the local robustness, while the latter has already been quantified in the first stage via KDE.

Stage 3: Generate Test Cases Around Test Seeds Considering the Local Distribution and Prediction Loss of AEs When searching for AEs locally around a test seed given by the 2nd stage, we develop a two-step GA in which the objective function is defined as a *fusion* of the prediction loss (cf. Definition 3) and the local distribution (modelled by common perceptual quality metrics). Such fusion of two fitness functions allows the trade-off between the "strength of being adversarial" and the perceptual quality of the detected AEs. The optimisation is subject to the constraint of only exploring in a norm ball whose central point is the test seed and with a radius smaller than the r-separation distance (cf. Remark 1).

While our chosen technical solutions are effective and popular, alternatives may also suffice for the purpose of each stage.

4.3.2 Approximation of the Global Distribution

Given the training dataset $\mathcal{D}$, the task of approximating the input distribution is equivalent to estimating a Probability Density Function (PDF) over the input domain $\mathcal{X}$ given $\mathcal{D}$. Despite this is a common problem with many established solutions, it is hard to accurately approximate the distribution due to the relatively sparse data of $\mathcal{D}$, compared to the high dimensionality of the input domain $\mathcal{X}$. So the practical solution is to do dimensionality reduction and then estimate the global distribution, which indeed is the first step of all existing methods of distribution-aware DL testing.

Specifically, we choose VAE-Encoder+KDE[2] for their simplicity and popularity. Assume $\mathcal{D}$ contains n samples and each $x_i \in \mathcal{D}$ is encoded by VAE-Encoder as a Gaussian distribution

[2]We only use the encoder of VAEs for feature extraction, rather than generate new data from the decoder, which is different to other methods mentioned in Section 4.2.2.

z_i in the latent space, we can estimate the PDF of z (denoted as $Pr(z)$) based on the encoded $\mathcal{D}$. The $Pr(z)$ conforms to the mixture of Gaussian distributions, i.e., $z \sim \mathcal{N}(\mu_{z_i}, \sigma_{z_i})$. Notably, this mixture of Gaussian distributions nicely aligns with the gist of adaptive KDE [135], which uses the following estimator:

$$p_g(x \in \eta_z) \propto Pr(z) \simeq \frac{1}{n} \sum_{i=1}^{n} K_{h_i}(z - \mu_{z_i}) \tag{4.5}$$

That is, when choosing a Gaussian kernel for K in Eqn. (4.5) and adaptively setting the bandwidth parameter $h_i = \sigma_{z_i}$ (i.e., the standard deviation of the Gaussian distribution representing the compressed sample z_i), the VAE-Encoder and KDE are combined "seamlessly". Finally, our global distribution $p_g(x \in \eta_z)$ (a pooled probability of all inputs in the region η_z that corresponds to a point z in the latent space) is proportional to the approximated distribution of z with the PDF $Pr(z)$.

Running Example: The left diagram in Fig. 4.2 depicts the global distribution learnt by KDE, after projected to a two-dimensional space for visualisation. The peaks[3] are evaluated with highest probability density over the latent space by KDE.

4.3.3 Test Seeds Selection

Selecting test seeds is actually about choosing which norm balls (around the test seeds) to test for AEs. To be cost-effective, we want to test those with higher global probabilities and lower local robustness at the same time. For the latter requirement, there is potentially a paradox:

Remark 3 (A Paradox of Selecting Unrobust Norm Balls)**.** *To be informative on which norm balls to test for AEs, we need to estimate the local robustness of candidate norm balls (by invoking robustness estimators to quantify $\mathcal{R}_l(\eta, y)$, e.g., [115]). However, local robustness evaluation itself is usually about sampling for AEs (then fed into statistical estimators) that consumes the testing resources.*

To this end, instead of *directly evaluating* the local robustness of a norm ball, we can only *indirectly predict* it (i.e., without testing/searching for AEs) via auxiliary information that we call local robustness indicators (cf. Definition 2). In doing so, we save all the testing budget for the later stage when generating local test cases.

Given a test seed x with label y, we propose two robustness indicators (both relate to the vulnerability of the test seed to adversarial attacks)—the *prediction gradient based score* (denoted as S_{grad}) and the *score based on separation distance of the output-layer activation* (denoted as S_{sep}):

$$\begin{aligned} S_{grad} &= ||\nabla_x \mathcal{J}(f(x), y)||_\infty \\ S_{sep} &= \min_{\hat{x}} ||f(x) - f(\hat{x})||_\infty \quad \text{s.t. } y \neq \hat{y} \end{aligned} \tag{4.6}$$

[3]Most training data lie in this region or gather around the region.

These allow prediction of a whole norm ball's local robustness by the limited information of its central point (the test seed). The gradient of a DNN's prediction with respect to the input is a white-box metric, that widely used in adversarial attacks, such as FGSM [1] and PGD [24] attacks. A greater gradient calculated at a test seed implies that AEs are more likely to be found around it. The activation separation distance is regarded as a black-box metric and refers to the minimum L_∞ norm between the output activations of the test seed and any other data with different labels. Intuitively, a smaller separation distance implies a greater vulnerability of the seed to adversarial attacks. We later show empirically that indeed these two indicators are highly correlated with the local robustness.

After quantifying the two required factors, we combine them in a way that was inspired by [136]. In [136], the DL reliability metric is formalised as a weighted sum of local robustness where the weights are operational probabilities of local regions. To align with that reliability metric, we do the following **steps to select test seeds**:

(i) For each data-point x_i in the test set, we calculate its global probability (i.e., $Pr(z_i)$ where z_i is its compressed point in the VAE latent space) and one of the local robustness indicators (either white-box or black-box, depending on the available information).

(ii) Normalise both quantities to a same scale.

(iii) Rank all data-points by the product of their global probability and local robustness indicator.

(iv) Finally we select top-k data-points as our test seeds, and k depends on the testing budget.

Running Example: In the middle diagram of Fig. 4.2, we add in the local robustness indicator results of the training data which are represented by a scale of colours—darker means lower predicted local robustness while lighter means higher predicated local robustness. By our method, test seeds selected are both from the highest peak (high probability density area of the global distribution) and relatively darker ones (lower predicated local robustness).

4.3.4 Local Test Cases Generation

Not all AEs are equal in terms of the "strength of being adversarial", and stronger AEs are associated with higher prediction loss (cf. Definition 3). Detecting AEs with higher prediction loss may benefit more when considering the future "debuging" step, e.g., by adversarial retraining [124]. Thus, at this stage, we want to search for AEs that are both "strongly being adversarial" and "less likely to be noticed by humans". That is, the local test case generation can be formulated as the following optimisation given a seed (x, y):

$$\begin{aligned} &\max_{x'} \mathcal{J}(f(x'), y) + \alpha \cdot p_l(x' | x' \in \eta_{z_x}) \\ &\text{s.t.} \, ||x - x'||_\infty \leq r \end{aligned} \tag{4.7}$$

where $\mathcal{J}$ is the prediction loss, $p_l(x'|x' \in \eta_{z_x})$ is the local distribution (note, z_x represents the latent features of test seed x), r is the r-separation distance, and α is a coefficient to balance the two terms. As what follows, we note two points on Eqn. (4.7): why we need the constraint and how we quantify the local distribution.

The constraint in Eqn. (4.7) determines the right *locality* of local robustness—the "neighbours" that should have the same ground truth label y as the test seed. We notice the r-separation property of real-world image datasets (cf. Remark 1) provides a sound basis to the question. Thus, it is formalised as a constraint that the optimiser can only search in a norm ball with a radius smaller than r, to guarantee the detected AEs are indeed "adversarial" to label y.

While the feature level information is captured by the global distribution over a latent space, we only consider how the pixel level information is distributed in terms of *perceptual quality* to humans. Three common quantitative metrics—MSE, PSNR and SSIM introduced in Section 4.2.3—are investigated. We note, those three metrics by no means are the true local distribution representing perceptual quality, rather quantifiable indicators from different aspects. Thus, in the optimisation problem of Eqn. (4.7), replacing the local distribution term with them would suffice our purpose. So, we redefine the optimisation problem as:

$$\max_{x'} \mathcal{J}(f(x'), y) + \alpha \cdot \mathcal{L}(x, x'), \quad \text{s.t. } ||x - x'||_\infty \leq r \tag{4.8}$$

where $\mathcal{L}(x, x')$ represents those perceptual quality metrics correlated with the local distribution of the seed x. Certainly, implementing $\mathcal{L}(x, x')$ requires some prepossessing, e.g., normalisation and negation, depending on which metric is adopted.

Considering that the second term of the objective function in Eqn. (4.8) may not be differentiable and/or the DL model's parameters are not always accessible, we propose a black-box approach to solve the optimisation problem that generates local test cases. It is based on a GA with two fitness functions to effectively and efficiently detect AEs, as shown in Algorithm 3.

Algorithm 3 presents the **process of generating a set of m test cases $\mathcal{T}$** from a given seed x with label y (denoted as (x, y)). At line 1, we define two fitness functions (the reason behind it will be discussed next). We initialise the population by adding uniform noise in range $(-r, +r)$ to the test seed, at line 2-4. At line 5-16, the population is iteratively updated by evaluating the fitness functions, selecting the parents, conducting crossover and mutation. At line 17-20, the best m fitted individuals in population are chosen as test cases. The crossover and mutation are regular operations in GA based test cases generation for DL models [116], while two fitness functions work alternatively to guide the selection of parents.

The reason why we propose two fitness functions is because, we notice that there is a trade-off between the two objectives $\mathcal{J}$ and $\mathcal{L}$ in the optimisation. Prediction loss $\mathcal{J}$ is related to the adversarial strength, while $\mathcal{L}$ indicates the local distribution. Intuitively, generating the test cases with high local probability tends to add small amount of perturbations to the seed, while a greater perturbation is more likely to induce high prediction loss. To avoid the competition between the two terms that may finally leads to a failure of detecting AEs, we define two fitness functions to precisely control the preference at different stages:

$$F_1 = \mathcal{J}(f(x'), y), \quad F_2 = \mathcal{J}(f(x'), y) + \alpha \cdot \mathcal{L}(x, x') \tag{4.9}$$

Algorithm 3: Two-Step GA Based Local Test Cases Generation

```
Input: Test seed $(x, y)$, neural network function $f(x)$, local perceptual quality
       metric $\mathcal{L}(x, x')$, population size $N$, maximum iterations $T$, norm ball radius
       $r$, weight parameter $\alpha$.
Output: A set of $m$ test cases $\mathcal{T}$
1  $F_1 = \mathcal{J}(f(x'), y)$, $F_2 = \mathcal{J}(f(x'), y) + \alpha \cdot \mathcal{L}(x, x')$
2  for $i = 1, ..., N$ do
3      $\mathcal{T}[i] = x + uniform(-r, +r)$
4  while $t < T$ or $\max(fit\_list_2)$ does not converge do
5      $fit\_list_1 = cal\_fitness(F_1, \mathcal{T})$
6      $fit\_list_2 = cal\_fitness(F_2, \mathcal{T})$
7      if $majority(fit\_list_1 < 0)$ then
8          $parents = selection(fit\_list_1, \mathcal{T})$
9      else
10         $parents = selection(fit\_list_2, \mathcal{T})$
11     $\mathcal{T} = crossover(parents, N)$
12     $\mathcal{T} = mutation(\mathcal{T}) \cup parents$
13     $t = t + 1$
14 $fit\_list_2 = cal\_fitness(F_2, \mathcal{T})$
15 $idx = \text{argmax}(fit\_list_2)[: m]$
16 $\mathcal{T} = \mathcal{T}[idx]$
17 return test set $\mathcal{T}$
```

At early stage, F_1 is optimised to quickly direct the generation of AEs, with $F_1 > 0$ meaning the detection of an AE. When most individuals in the population are AEs, i.e., $majority(fit\_list_1 \geq 0)$, the optimisation moves to the second stage, in which F_2 is replaced by F_1 to optimise the local distribution indicator as well as the prediction loss. It is possible[4] that the prediction loss of most individuals again become negative, then the optimisation will go back to the first stage. With such a mechanism of alternatively using two fitness functions in the optimisation, the proportion of AEs in the population is effectively prevented from decreasing.

Algorithm 3 describes the process for generating m local test cases given a single test seed. Suppose n test seeds are selected earlier and in total M local test cases are affordable, we can allocate, for each test seed x_i, the number of local test cases m_i, according to the n (re-normalised) global probabilities, which emphasises more on the role of distribution in our detected set of AEs.

Running Example: The right diagram in Fig. 4.2 plots the local distribution using MSE as its indicator, and visualises the detected AEs by different testing methods. Unsurprisingly, all AEs detected by our proposed HDA testing are located at the high density regions (and very close to the central test seed), given it considers the perceptual quality metric as one of the optimisation objectives in the two-step GA based test case generation. In contrast, other methods (PGD and coverage-guided) are less effective.

4.4 Evaluation

We evaluate the proposed HDA method by performing extensive experiments to address the following research questions (RQs):

RQ1 (Effectiveness): How effective are the methods adopted in the three main stages of HDA? Namely, we conduct experiments to *i)* examine the accuracy of combining VAE-Encoder+KDE to approximate the global distribution; *ii)* check the correlation significance of the two proposed local robustness indicators with the local robustness; *iii)* investigate the effectiveness of our two-step GA for local test cases generation.

RQ2 (AE Quality): How is the quality of AEs detected by HDA? Comparing to conventional attack-based and coverage-guided methods and more recent distribution-aware testing methods of OODA and FODA, we introduce a comprehensive set of metrics to evaluate the quality of AEs detected by HDA and others.

RQ3 (Sensitivity): How sensitive is HDA to the DL models under testing? We carry out experiments to assess the capability of HDA applied on DL models (adversarially trained) with different levels of robustness.

[4]Especially when a large α is used, i.e., with preference on detecting AEs with high local probability than with high adversarial strength, cf. the aforementioned trade-off.

RQ4 (Robustness Growth): How useful is HDA to support robustness growth of the DL model under testing? We examine the global robustness of DL models after "fixing" the AEs detected by various testing methods.

4.4.1 Experiment Setup

In **RQ1** and **RQ2**, we consider five popular benchmark datasets and five diverse model architectures for evaluation. Details of the datasets and trained DL models under testing are listed in Table 4.1. The norm ball radius r is calculated based on the r-separation distance (cf. Remark 1) of each dataset. In **RQ3**, we add the comparison on DL models, enhanced by PGD-based adversarial training, for sensitivity analysis. Table 4.1 also records the accuracy of these adversarially trained models. Adversarial training trades the generalisation accuracy for the robustness as expected (thus a noticeable decrement of the training and testing accuracy) [127]. In **RQ4**, we firstly sample 10000 data points from the global distribution as validation set and detect AEs around them by different methods. Then, we fine-tune the normally trained models with training dataset augmented by these AEs. 10 epochs are taken along with 'slow start, fast decay' learning rate schedule [137] to reduce the computational cost while improve the accuracy-drop and robustness. To empirically estimate the global robustness on validation set, we find another set of AEs according to local distribution, different from the fine-tuning data. These validating AEs are miss-classified by normally trained models. Thus, empirical global robustness of normally trained models is set to 0 as the baseline.

For readers' convenience, all the metrics used in **RQ2**, **RQ3** and **RQ4** for comparisons are listed in Table 4.2. The metrics are introduced to comprehensively evaluate the quality of detected AEs and the DL models from different aspects.

Table 4.1: Details of the datasets and DL models under testing.

Dataset	Image Size	r	DL Model	Normal Training		Adversarial Training	
				Train Acc.	Test Acc.	Train Acc.	Test Acc.
MNIST	$1 \times 32 \times 32$	0.1	LeNet5	1.000	0.991	0.992	0.988
Fashion-MNIST	$1 \times 32 \times 32$	0.08	AlexNet	0.952	0.910	0.899	0.882
SVHN	$3 \times 32 \times 32$	0.03	VGG11	0.945	0.944	0.882	0.889
CIFAR-10	$3 \times 32 \times 32$	0.03	ResNet20	0.994	0.900	0.748	0.724
CelebA	$3 \times 64 \times 64$	0.05	MobileNetV1	0.953	0.918	0.877	0.853

All experiments were run on a machine of Ubuntu 18.04.5 LTS x86_64 with Nvidia A100 GPU and 40G RAM. The source code, DL models, datasets and all experiment results are publicly available at `https://github.com/havelhuang/HDA-Testing`.

Table 4.2: Evaluation metrics for the quality of detected AEs and DL models

Metrics	Meanings
AE Prop.	Proportion of AEs in the set of test cases generated from selected test seeds
Pred. Loss	Adversarial strength of AEs as formally defined by Definition 3
p_g	Normalised global probability density of test-seeds/AEs
$\mathcal{R}_l$	Local robustness to the correct classification label, as formally defined by Definition 1
$\hat{\mathcal{R}}_g$	Empirical global robustness of DL models over input domain as defined in Definition 5
FID	Distribution difference between original images (test seeds) and perturbed images (AEs)
ϵ	Average perturbation distance between test seeds and AEs
% of Valid AEs	Percentage of "in-distirbution" AEs in all detected AEs

4.4.2 Evaluation Results and Discussions

RQ1

There are 3 sets of experiments in **RQ1** to examine the accuracy of technical solutions in our tool-chain, corresponding to the 3 main stages respectively.

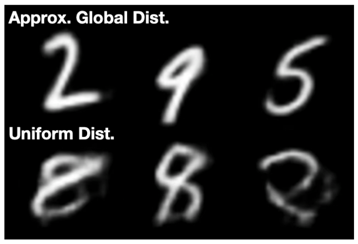
First, to approximate the global distribution, we essentially proceed in two steps—dimensionality reduction and PDF fitting, for which we adopt the VAE-Encoder+KDE solution. Notably, the VAE trained in this step is for data-compression only (not for generating new data). To reflect the effectiveness of both aforementioned steps, we (i) compare VAE-Encoder with the Principal Component Analysis (PCA), and (ii) then measure the FID between the training dataset and a set of random samples drawn from the fitted global distribution by KDE.

PCA is a common approach for dimensionality reduction. We compare the performance of VAE-Encoder and PCA from the following two perspectives. The *quality of latent representation* can be measured by the *clustering* and *reconstruction accuracy.* To learn the global distribution from latent data, we require that latent representations should group together data-points based on semantic features and can be decoded to reconstruct the original images with less information loss. Therefore, we apply K-means clustering to the latent data and calculate the Completeness Score (CS), Homogeneity Score (HS) and V-measure Score (VS) [138] for measuring the ability of clustering. While, the reconstruction loss is calculated based on the MSE. As is shown in Table 4.3, VAE-Encoder achieves higher CS, HS, VS scores and less reconstruction loss than PCA. In other words, the latent representations encoded by VAE-Encoder is more significant in terms of capturing features than that of PCA.

To evaluate the accuracy of using KDE to fit the global distribution, we calculate the FID between a new dataset (with 1000 samples) based on the fitted global distribution by KDE and the training dataset. The FID scores are shown in Table 4.4. As a baseline, we also present the results of using a uniform distribution over the latent space. As expected, we observe that all FID scores based on approximated distributions are significantly smaller (better). We further decode the newly generated dataset for visualisation in Fig. 4.3, from which we see the generated images by KDE keep high fidelity while the uniformly sampled images are not human-perceptible.

Table 4.3: Quality of Latent Representation in PCA & VAE-Encoder

Dataset	PCA				VAE-Encoder			
	Clustering			Recon. Loss	Clustering			Recon. Loss
	CS	HS	VS		CS	HS	VS	
MNIST	0.505	0.508	0.507	44.09	0.564	0.566	0.565	27.13
F.-MNIST	0.497	0.520	0.508	55.56	0.586	0.601	0.594	23.72
SVHN	0.007	0.007	0.007	65.75	0.013	0.012	0.013	66.21
CIFAR-10	0.084	0.085	0.085	188.22	0.105	0.105	0.105	168.44
CelebA	0.112	0.092	0.101	764.94	0.185	0.150	0.166	590.54



Dataset	Global Dist.	Uni. Dist.
MNIST	**0.395**	13.745
Fashion-MNIST	**0.936**	90.235
SVHN	**0.875**	143.119
CIFAR-10	**0.285**	12.053
CelebA	**0.231**	8.907

Figure 4.3 & Table 4.4: Samples drawn from the approximated global distribution by KDE and a uniform distribution over the latent feature space (Figure); and FID to the ground truth based on 1000 samples (Table).

Answer to **RQ1** on HDA stage 1: The combination of VAE-Encoder+KDE may accurately approximate the global distribution.

Move on to stage 2, we study the correlations between a norm ball's local robustness and its two indicators proposed earlier—the prediction gradient based score and the score based on separation distance of output-layer activation (cf. Eq. 4.6).

We invoke the tool [115] for estimating the local robustness $\mathcal{R}_l$ defined in Definition 1. Based on 1000 randomly selected data-points from the test set as the central point of 1000 norm balls, we calculate the local robustness of each norm ball[5] as well as the two proposed indicators. Then, we do the scatter plots (in log-log scale[6]), as shown in Fig. 4.4. Apparently, for all 5 datasets, the indicator based on activation separation distance is negatively correlated (1st row), while the gradient indicator is positively correlated with the estimated local robustness (2nd row). We further quantify the correlation by calculating the Pearson coefficients, as recorded in Table 4.5. We observe, both indicators are highly correlated with the local robustness, while the gradient based indicator is stronger. This is unsurprising, because the activation separation distance is a black-box metric which is

[5]Radius r is usually small by definition (cf. Remark 1), yielding very small $log(1 - \mathcal{R}_l)$.

[6]There are dots collapsed on the vertical line of $log(1 - R) = -70$, due to a limitation of the estimator [115]—it terminates with the specified threshold when the estimation is lower than that value. Note, the correlation calculated with such noise is not undermining our conclusion, rather the real correlation would be even higher.

usually weaker than the white-box gradient information.

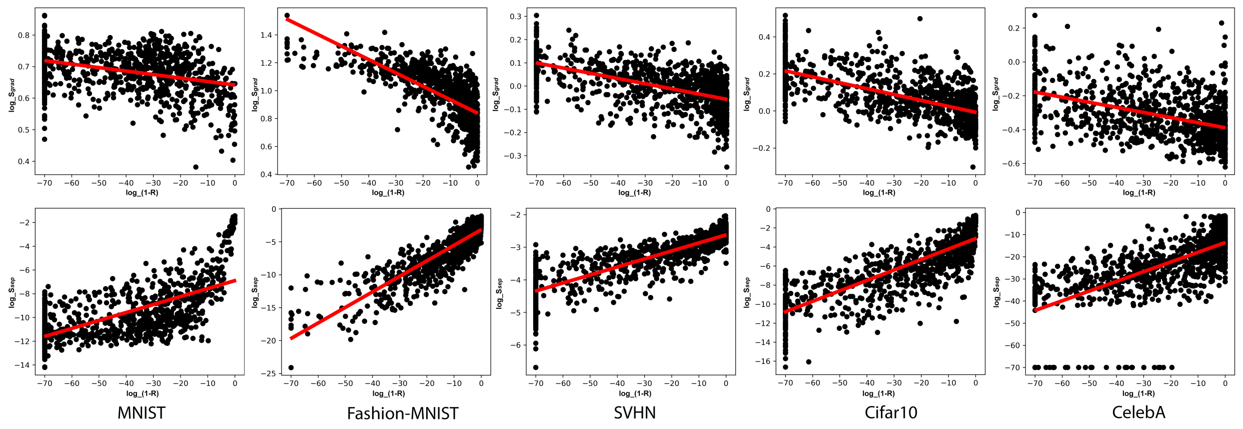


Figure 4.4: Scatter plots of the local robustness evaluation vs. its two indicators, based on 1000 random norm balls.

Table 4.5: Pearson correlation coefficients (in absolute values) between the local robustness & its two indicators.

Dataset	S_{grad}	S_{sep}
MNIST	0.672	0.379
Fashion-MNIST	0.872	0.716
SVHN	0.848	0.612
CIFAR-10	0.832	0.646
CelebA	0.699	0.468

Answer to **RQ1** on HDA stage 2: The two proposed local robustness indicators are significantly correlated with the local robustness.

For the local test case generation in stage 3, by configuring the parameter α in our two-step GA, we may do trade-off between the "strength of being adversrial" (measured by prediction loss $\mathcal{J}$) and the perceptual quality (measured by a specific $\mathcal{L}$), so that the quality of detected AEs can be optimised.

In Fig. 4.5, we visualise the changes of the two fitness values as the iterations of the GA. As shown in the first plot, only the prediction loss $\mathcal{J}$ is taken as the objective function (i.e., $\alpha = 0$) during the whole iteration process. The GA can effectively find AEs with maximised adversarial strength, which is observed from that the prediction loss of best fitted test case in the population converges after hundreds of iterations. From the second to the last plot, the other fitness function $\mathcal{L}$ representing the local distribution information is added to the objective function (i.e., $\alpha > 0$), they are MSE, PSNR and SSIM. Intuitively, higher local probability density implies smaller MSE and greater PSNR and SSIM.

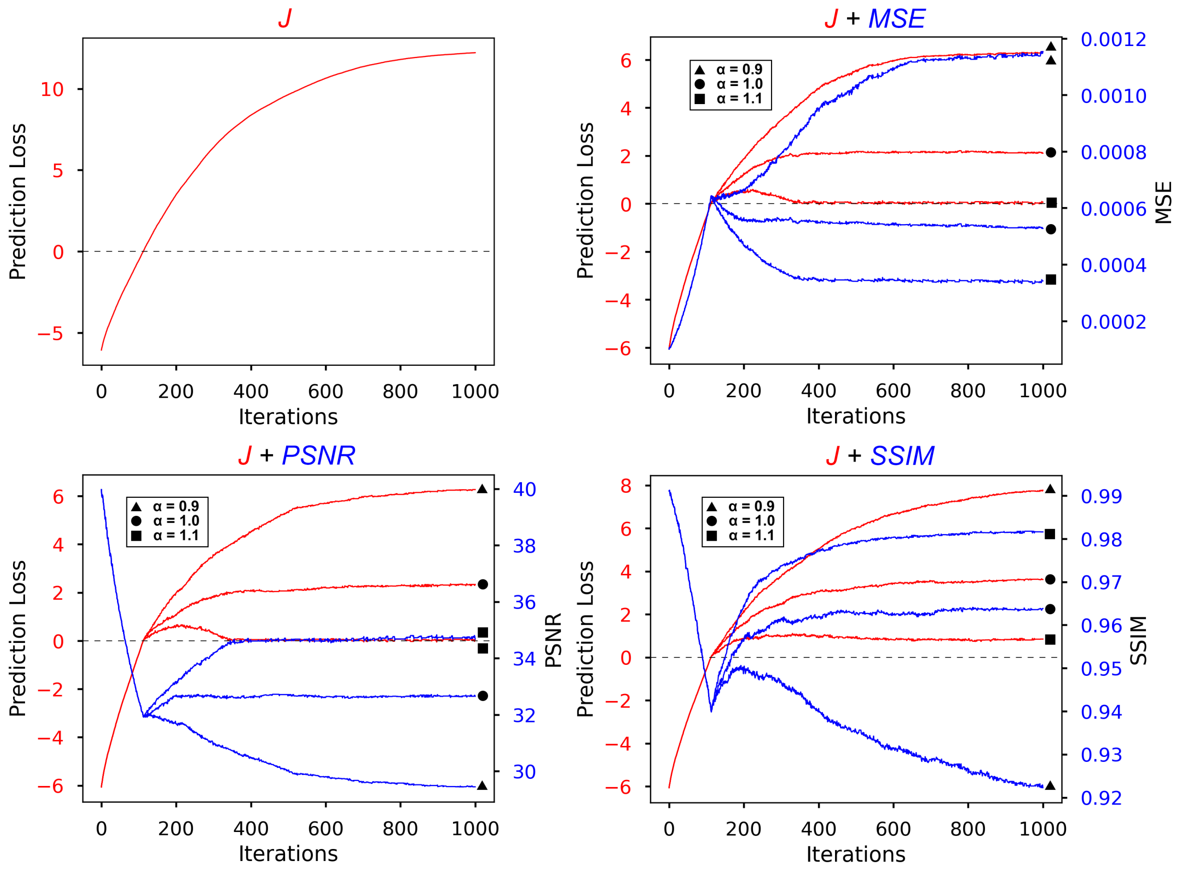


Figure 4.5: The prediction loss (red) and the three quantified local distribution indicators (blue) of the best fitted test case during the iterations of our two-step GA based local test case generation.

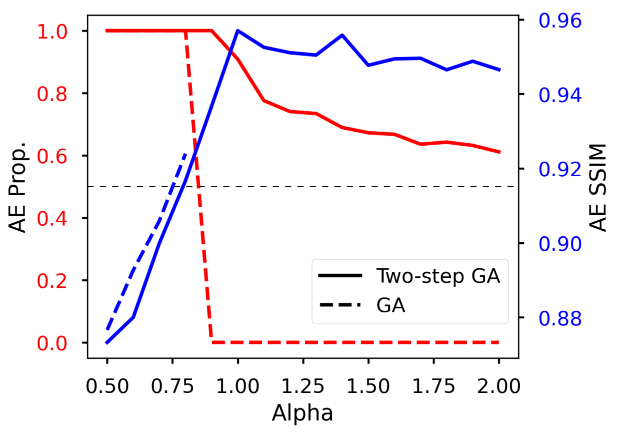


Figure 4.6: Comparison between regular GA and two-step GA.

Thanks to the two-step setting of the fitness functions, the prediction loss $\mathcal{J}$ of best fitted test case goes over 0 quickly in less than 200 iterations, which means it detects a first AE in the population. The $\mathcal{J}$ of the best fitted test case is always quite close to the rest in the population, thus we may confidently claim that many AEs are efficiently detected by the population not long after the first AE was detected. Then, the optimisation goes to the second stage, in which the quantified local distribution indicator $\mathcal{L}$ is pursued. The $\mathcal{J}$ and $\mathcal{L}$ finally converge and achieve a balance between them. If we configure the coefficient α, the balance point will change correspondingly. A greater α (e.g., $\alpha = 1.1$ in the plots) detects more natural AEs (i.e., with higher local probability density), and the price paid is that the detected AEs are with weaker adversarial strength (i.e., with smaller but still *positive* prediction loss).

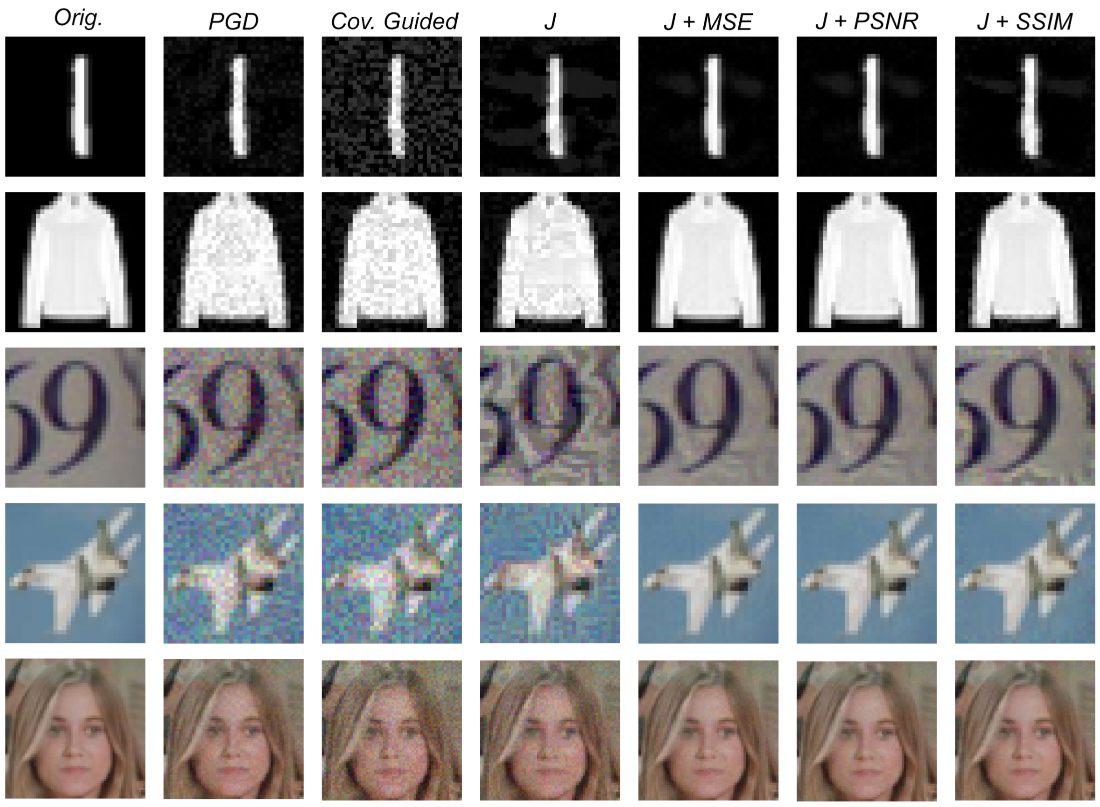


Figure 4.7: AEs detected by our two-step GA (last 3 columns) & other methods

We further investigate the advantages of our 2-step GA over the regular GA (using F_2 as the objective function). In Fig. 4.6, as α increases, the proportion of AEs in the population exhibits a sharp drop to 0 when using the regular GA. In contrast, the two-step GA prevents such decreasing of the AE proportion while preserving it at a high-level of 0.6, even when α is quite large. Moreover, larger α represents the situations when the AEs are more natural—

as shown by the blue curves[7], the local distribution indicator (SSIM in this case) is only sufficiently high when α is big enough. Thus, compared to the regular GA, we may claim our novel 2-step GA is more robust (in detecting AEs) to the choices of α and more suitable in our framework for detecting AEs with high local probabilities.

Fig. 4.7 displays some selected AEs from the five datasets. Same as the PGD and the coverage-guided testing, if we only use the prediction loss $\mathcal{J}$ as the objective function in the GA, the perturbations added to the images can be easily told. In stark contrast, AEs generated by our two-step GA (with the 3 perceptual quality metrics in the last 3 columns) are of high quality[8] and indistinguishable with human-eyes from the original images (first column).

Answer to **RQ1** on HDA stage 3: Two-step GA based local test case generation can effectively detect AEs with high perception quality.

RQ2

We compare our HDA with state-of-the-art AE detection methods in two sets of experiments. In the first set, we focus on comparing with the adversarial attack and coverage-guided testing (i.e., the typical PGD attack and neuron coverage metric for brevity, while the conclusion can be generalised to other attacks and coverage metrics). Then in the second set of experiments, we show the advantages of our HDA over other distribution-aware testing methods.

In fact, both PGD attack and coverage-guided testing do not contribute to test seeds selection. They simply use randomly sampled data from the test set as test seeds, by default. Thus, we only need to compare the randomly selected test seeds with our "global distribution probability[9] plus local robustness indicated" test seeds, shown as "'$p_g + \mathcal{R}_l$" in Table 4.6. Specifically, for each test seed, we calculate two metrics—the local robustness $\mathcal{R}_l$ of its norm ball and its corresponding global probability p_g. We invoke the estimator of [115] to calculate the former ($log(1 - \mathcal{R}_l)$, to be exact). To reduce the sampling noise, we repeat the test seed selection 100 times and present the averaged results in Table 4.6.

From Table 4.6, we observe: (i) test seeds selected by our method have much higher global probability density, meaning their norm balls are much more representative of the data distribution; (ii) the norm balls of our test seeds have worse local robustness, meaning it is more *cost-effective* to detect AEs in them. These are unsurprising, because we have explicitly considered the distribution and local robustness information in the test seed selection.

Finally, the overall evaluation on the generated test cases and the detected AEs by them are shown in Table 4.7. The results are presented in two dimensions—3 types of testing methods versus 2 ways of selecting test seeds, yielding 6 combinations (although by default,

[7]The blue dashed line stops earlier as there is no AEs in the population when α is big.

[8]All 3 perceptual quality metrics perform good in grey-scale images, while SSIM performs the best in colourful images and tends to add noise to the background.

[9]Refer to Section 4.3.2 for the calculation. The value of probability density is further normalised by training dataset for a better presentation.

Table 4.6: Comparison between randomly selected test seeds and our "$p_g + \mathcal{R}_l$ indicated" test seeds (averaging over 100 test seeds).

Dataset	Random Test Seeds		$p_g + \mathcal{R}_l$ Test Seeds	
	$log(1 - \mathcal{R}_l)$	p_g	$log(1 - \mathcal{R}_l)$	p_g
MNIST	-48.7	0.0049	-45.6	0.0835
Fashion-MNIST	-21.9	0.0074	-18.4	0.0632
SVHN	-22.1	0.0055	-21.2	0.0804
CIFAR-10	-23.3	0.0101	-19.8	0.3439
CelebA	-36.3	0.0069	-32.7	0.1272

PGD and Coverage-guided methods are using random seeds, while our method is using the "$p_g + \mathcal{R}_l$" seeds). For each combination, we study 4 metrics (cf. Table 4.2 for meanings behind them): (i) the AE proportion; (ii) the average prediction loss; (iii) the FID[10] of the test set quantifying the image quality; and (iv) the computational time (and an additional coverage rate for coverage-guided testing). We note the observations on these 4 metrics in the following paragraphs.

Table 4.7: Evaluation of the generated test cases and detected AEs by PGD Attack, coverage-guided testing and the proposed HDA testing (all results are averaged over 100 seeds)

Seed	Dataset	PGD Attack				Coverage Guided Testing					Hierarchical Distribution-Aware Testing			
		AE Prop.	Pred. Loss	FID	Time(s)	Cov. Rate	AE Prop.	Pred. Loss	FID	Time(s)	AE Prop.	Pred. Loss	FID	Time(s)
Random Seeds	MNIST	0.205	**7.01**	0.46	**0.48**	0.859	0.001	1.48	0.76	187.92	**0.600**	1.48	**0.16**	51.63
	F.-MNIST	0.957	**19.62**	1.89	**0.44**	0.936	0.228	2.15	3.65	131.47	**0.999**	6.08	**0.11**	63.36
	SVHN	0.866	**2.81**	95.81	**11.11**	0.976	0.004	0.09	98.49	343.47	**0.922**	2.37	**95.22**	156.46
	CIFAR-10	1.000	**39.74**	87.51	**11.22**	0.988	0.196	3.32	93.32	542.73	**1.000**	37.79	**75.59**	221.29
	CelebA	0.979	**119.95**	78.53	**12.64**	0.992	0.052	12.09	84.42	931.65	**1.000**	96.03	**69.39**	233.78
p_g+$\mathcal{R}_l$ Seeds	MNIST	0.986	**11.95**	0.21	**0.47**	0.873	0.076	1.37	0.82	187.73	**1.000**	3.59	**0.01**	53.05
	F.-MNIST	1.000	**26.24**	0.69	**0.44**	0.950	0.322	2.41	1.33	132.39	**1.000**	9.73	**0.03**	62.61
	SVHN	0.992	**2.91**	87.50	**11.44**	0.979	0.038	0.09	93.05	336.64	**1.000**	2.12	**83.02**	156.39
	CIFAR-10	1.000	**40.08**	83.35	**12.74**	0.989	0.221	3.98	87.05	543.32	**1.000**	33.45	**70.27**	221.38
	CelebA	0.998	**120.51**	74.83	**11.54**	0.988	0.067	8.72	80.49	939.78	**1.000**	93.48	**67.77**	233.97

Regarding the AE proportion in the set of generated test cases, the default setting of our proposed approach is clearly the best (with score 1) among the 6 combinations. Both our novel test seed selection and two-step GA local test case generation methods contribute to the win. This can be told from the decreased AE proportion when using random seeds in our method, but still the result is relatively higher than most combinations. PGD, as a white-box approach using the gradient information, is also quite efficient in detecting AEs, especially when paired with our new test seed selection method. On the other hand, coverage-guided testing is comparatively less effective in detecting AEs (even with high coverage rate), yet our test seed selection method can improve it.

As per the results of prediction loss, PGD, as a gradient-descent based attacking method, unsurprisingly finds the AEs with the largest prediction loss. With better test seeds considering local robustness indicators selected by our method, the prediction loss of PGD can be

[10]To show how close the perturbed test cases are to the test seeds in the latent space, we use the last convolutional layer of InceptionV3 to extract the latent representations of colour images for FID. InceptionV3 is a well-trained CNN and commonly used to show FID that captures the perturbation levels, e.g., in [134]. While InceptionV3 is used for colour images, VAE is used for grey-scale datasets MNIST and Fashion-MNIST.

even higher. Both coverage-guided and our HDA testing are detecting AEs with relatively lower prediction loss, meaning the AEs are with "weaker adversarial strength". The reason for the low prediction loss of AEs detected by our approach is that our two-step GA makes the trade-off and sacrifices it for AEs with higher local probabilities (i.e., more natural). This can be seen through the small FID of our test set. PGD, on the other hand, has relatively high FID scores, as well as coverage-guided testing.

On the computational overheads, we observe PGD is the most efficient, given it is by nature a white-box approach using the gradient-descent information. While, our approach is an end-to-end black-box approach (if without using the gradient based indicator when selecting test seeds) requiring less information and being more generic, at the price of being relatively less efficient. That said, the computational time of our approach is still acceptable and better than coverage-guided testing.

Answer to **RQ2** on comparing with adversarial attack and coverage-guided testing: HDA shows advantages over adversarial attack and coverage-guided testing on test seeds selection and generation of high perception quality AEs.

Next, we try to answer the difference between our HDA testing method and other distribution-aware testing as summarised earlier (the amber route of Fig. 4.1). We not only study the common evaluation metrics in earlier RQs, but also the input validation method in [30], which flags the validity of AEs according to a user-defined reconstruction probability threshold.

Table 4.8: Evaluation of AEs detected by OODA, FODA and our HDA testing methods (based on 100 test seeds).

Dataset	Tool	p_g	% of Valid AEs	ϵ	FID
MNIST	OODA	0.0055	29	0.81	2.29
	FODA	0.0030	100	0.73	0.47
	HDA	**0.0835**	**100**	**0.05**	**0.01**
SVHN	OODA	0.0046	21	0.82	128.84
	FODA	0.0021	100	0.31	110.73
	HDA	**0.0804**	**100**	**0.03**	**83.02**

As shown in Table 4.8, HDA can select test seeds from much higher density region on the global distribution and find more valid AEs than OODA. The reason behind this is that OODA aims at detecting outliers—only AEs have lower reconstruction probabilities (from the test seed) than the given threshold will be marked as invalid test cases. While, HDA explicitly explores the high density meanwhile error-prone regions by combining the global distribution and local robustness indicators. In other words, HDA does priority ordering (according to the global distribution and local robustness) and then selects the best, while OODA rules out the worst. As expected, FODA performs similarly bad as OODA in terms of p_g, since both are using randomly selected seeds. While, FODA has high proportion of valid AEs since the test cases are directly sampled from the distribution in latent space.

Regarding the perceptual quality of detected AEs, HDA can always find AEs with small pixel-level perturbations (ϵ) in consideration of the r-separation constraint, and with small FID thanks to the use of perceptual quality metrics (SSIM in this case) as objective functions. While OODA only utilises the reconstruction probability (from VAE) to choose AEs, and FODA directly samples test cases from VAE without any restrictions (thus may suffer from the oracle problem, cf. Remark 4 later). Due to the compression nature of generative models—they are good at extracting feature level information but ignore pixel level information [111], AEs detected by OODA and FODA are all distant to the original test seeds, yielding large ϵ and FID scores. Notably, the average distance ϵ between test seeds and AEs detected by OODA and FODA are much (7∼28 times) greater than the r-separation constraints (cf. Table 4.1), leading to the potential oracle issues of those AEs, for which we have the following remark:

Remark 4 (Oracle Issues of AEs Detected by OODA and FODA)**.** *AEs detected by OODA and FODA are normally distant to the test seeds with a perturbation distance even greater than the r-separation constraint. Consequently, there is the risk that the perturbed image may not share the same ground truth label of the test seed, and thus hard to determine the ground truth label of the "AE"*[11].

To visualise the difference between AEs detected by HDA, FODA and OODA, we present 4 examples in Fig. 4.8. We may observe the AEs detected by HDA are almost indistinguishable from the original images. Moreover, the AEs by FODA is a set of concrete evidence for Remark 4—it is actually quite hard to tell what is the ground truth label of some perturbed image (e.g., the bottom left one), while others appear to have a different label of the seed (e.g., the bottom right one should be with a label "1" instead of "7").

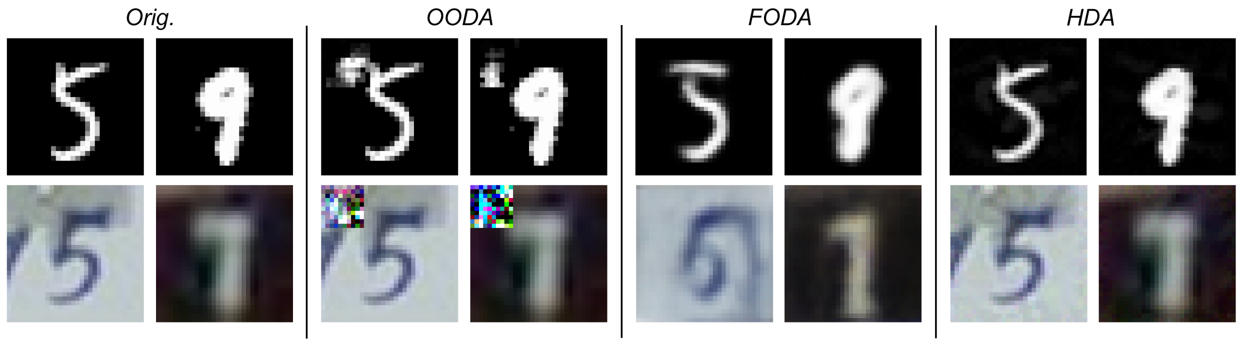


Figure 4.8: Example AEs detected by different distribution-aware testing methods.AEs detected by our HDA are indistinguishable from the original images, while AEs detected by FODA and OODA are of low perceptual quality and subject to the oracle issues noted by Remark 4.

[11]In quotes, because the perturbed image could be a "benign example" with a *correct* predicted label (but different to the test seed).

Answer to **RQ2** on comparing with other distribution-aware testing: Compared to OODA and FODA, the proposed HDA testing can detect more valid AEs, free of oracle issues, with higher global probabilities and perception quality.

RQ3

In earlier RQs, we have varied the datasets and model architectures to check the effectiveness of HDA. In this **RQ3**, we concern HDA's sensitivity to DL models with different levels of robustness. Adversarial training may greatly improve the robustness of DL models and is normally used as the defence to adversarial attack. To this end, we apply HDA on both normally and *adversarially trained* models (by [24] to be exact), and then compare with three most representative attacking methods—the most classic FGSM, the most popular PGD, and the most advanced one AutoAttack [139]. Experimental results are presented in Table 4.9.

Table 4.9: Evaluation of AEs generated by FGSM, PGD, AutoAttack and HDA on normally and adversarially trained DL models (all results are averaged over 100 test seeds).

Model	Dataset	FGSM			PGD			AutoAttack			HDA		
		p_g	AE Prop.	FID	p_g	AE Prop.	FID	p_g	AE Prop.	FID	p_g	AE Prop.	FID
Normally Trained	MNIST	0.0099	0.34	1.085	0.0099	0.53	0.639	0.0100	0.92	0.954	**0.0835**	**1.00**	**0.011**
	F.-MNIST	0.0109	0.78	3.964	0.0109	1.00	2.611	0.0109	1.00	4.505	**0.0635**	**1.00**	**0.013**
	SVHN	0.0041	0.77	114.65	0.0042	0.97	107.04	0.0042	0.99	108.41	**0.0804**	**1.00**	**79.21**
	CIFAR10	0.0115	0.93	112.32	0.0114	1.00	101.92	0.0115	1.00	108.15	**0.3442**	**1.00**	**67.13**
	CelebA	0.0090	0.81	99.226	0.0090	0.97	89.413	0.0091	1.00	91.591	**0.1285**	**1.00**	**67.71**
Adversarially Trained	MNIST	0.0100	0.09	0.993	0.0100	0.08	0.728	0.0105	0.11	0.634	**0.1944**	**0.62**	**0.049**
	F.-MNIST	0.0112	0.29	4.187	0.0112	0.34	3.492	0.0122	0.74	2.888	**0.0632**	**0.81**	**0.297**
	SVHN	0.0043	0.45	127.89	0.0040	0.50	121.25	0.0054	0.63	120.97	**0.0821**	**0.71**	**83.88**
	CIFAR10	0.0137	0.46	122.74	0.0136	0.50	118.55	0.0139	0.55	93.779	**0.3263**	**0.64**	**55.47**
	CelebA	0.0096	0.37	108.56	0.0095	0.39	105.96	0.0097	0.43	106.72	**0.2007**	**0.49**	**71.06**

As expected, after the adversarial training by [24], the robustness of all five DL models are greatly improved. This can be observed from the metric of AE Prop.: For all four methods, the proportion of AEs detected in the set of test case is sharply decreased for adversarially trained models, while HDA still outperforms others. Since the rationales behind the three adversarial attacks are without considering the input data distribution nor perception quality, it is unsurprising that their two sets of results for normally and adversarially trained models are quite similar, in terms of the metrics p_g and FID. On the other hand, the FID scores of AEs detected by HDA get worse but still better than others. The measured p_g on AEs detected by HDA also changed due to the variations of robustness indicators before and after the adversarial training, and yet much higher than all attacking methods.

Answer to **RQ3**: HDA is shown to be capable and superior to common adversarial attacks when applied on DL models with different levels of robustness.

RQ4

The ultimate goal of developing HDA testing is to improve the global robustness of DL models. To this end, we refer to a validation set of 10000 test seeds. We fine-tune [137] the DL models with AEs detected for validation set from different methods. Then, we calculate the train accuracy, test accuracy and empirical global robustness before and after the adversarial fine-tuning. Empirical global robustness is measured on a new set of on-distribution AEs for validation set, different from the fine-tuning data. Fine-tuning requires to know the ground truth label of the AEs, which cannot be satisfied due to the potential oracle issues of OODA and FODA (cf. Remark 4). Thus, we omit the comparison with OODA and FODA, while other results are presented in Table 4.10.

Table 4.10: Evaluation of DL models' train accuracy, test accuracy, and empirical global robustness (based on 10000 on-distribution AEs) after adversarial fine-tuning.

Dataset	No. of	PGD Attack			HDA Testing			Coverage Guided Testing		
	Test Cases	Train Acc.	Test Acc.	$\mathcal{R}_g$	Train Acc.	Test Acc.	$\mathcal{R}_g$	Train Acc.	Test Acc.	$\mathcal{R}_g$
MNIST	300	98.26%	97.64%	49.27%	99.98%	98.77%	**90.94%**	100.00%	98.93%	34.83%
	3000	99.10%	98.05%	84.09%	99.95%	98.72%	**99.28%**	100.00%	98.99%	47.91%
	30000	99.94%	98.65%	99.88%	100.00%	98.88%	**100.00%**	100.00%	98.77%	71.10%
F.-MNIST	300	97.41%	91.27%	68.04%	98.92%	91.35%	**70.00%**	97.62%	90.93%	47.05%
	3000	89.48%	87.34%	88.78%	94.49%	89.96%	**92.39%**	88.06%	84.94%	63.30%
	30000	86.67%	85.06%	95.00%	93.54%	89.70%	**97.89%**	88.60%	85.56%	84.71%
SVHN	300	95.01%	93.63%	48.84%	89.26%	87.78%	**62.93%**	97.25%	90.95%	16.79%
	3000	88.81%	88.94%	75.83%	92.96%	92.66%	**83.78%**	87.21%	84.06%	37.59%
	30000	78.72%	80.14%	80.14%	92.81%	91.91%	**94.91%**	87.32%	84.18%	66.58%
CIFAR-10	300	92.39%	85.60%	46.88%	93.38%	86.78%	**48.96%**	95.56%	86.22%	0.03%
	3000	88.78%	84.10%	76.46%	92.07%	86.42%	**92.92%**	93.22%	84.40%	0.48%
	30000	88.26%	82.99%	94.47%	91.62%	86.13%	**97.58%**	93.46%	84.30%	13.43%

We first observe that adversarial fine-tuning is effective to improve the DL models' empirical global robustness, measured by the prediction accuracy on AEs for normally trained models, while compromising the train/test accuracy as expected (in contrast to normal training in Table 4.1). In most cases, DL models enhanced by HDA testing suffers least from the drop of generalisation. The reason behind this is that HDA testing targets at AEs from high density regions on distributions, usually with small prediction loss, shown in Fig. 4.5. Thus, eliminating AEs detected by HDA testing requires relatively minor adjustment to DL's models, the generalisation of which can be easily tampered during the fine-tuning with new samples.

In terms of empirical global robustness, HDA testing detects AEs around test seeds from the high global distribution region, which are more significant to the global robustness improvement. It can be seen that with 3000 test cases generated by utilising 1000 test seeds, the HDA testing can improve empirical global robustness to nearly or over 90%, very closed to the fine-tuning with 30000 test cases from 10000 test seeds. This means the distribution-based test seeds selection is more efficient than random test seeds selection. Moreover, even fine-tuning with 30000 test cases, leveraging all the test seeds in the validation set, HDA is still better than PGD attack and coverage-guided testing, due to the consideration of local distributions (approximated by naturalness). We notice that PGD-based adversarial fine-tuning minimises the maximum prediction loss within the local region, which is also effective to eliminate the natural AEs, but sacrificing more train/test accuracy. DL models fined-tuned with HDA testing achieve the best balance between the generalisation and global robustness.

Answer to **RQ4**: Compared with adversarial attack and coverage-guide testing, HDA contributes more to the growth of global robustness, while mitigating the drop of train/test accuracy during adversarial fine-tuning.

4.5 Threats to Validity

4.5.1 Internal Validity

Threats may arise due to bias in establishing cause-effect relationships, simplifications and assumptions made in our experiments. In what follows, we list the main threats of each research question and discuss how we mitigate them.

Threats from HDA Techniques

In **RQ1**, both the performance of the VAE-Encoder and KDE are threats. For the former, it is mitigated by using four established quality metrics (in Table 4.3) on evaluating dimensionality reduction techniques and compared to the common PCA method. It is known that KDE performs poorly with high-dimensional data and works well when the data dimension is modest [140, 141]. The data dimensions in our experiments are relatively low given the datasets have been compressed by VAE-Encoder, which mitigates the second threat. When studying the local robustness indicators, quantifying both the indicators and the local robustness may subject to errors, for which we reduce them by carefully inspecting the correctness of the script on calculating the indicators and invoking a reliable local robustness estimator [115] with fine-tuned hyper-parameters. For using two-step GA to generate local test cases, a threat arises by the calculation of norm ball radius, which has been mitigated by r-separation distance presented in the paper [123]. Also, the threat related to estimating the local distribution is mitigated by quantifying its three indicators (MSE, PSNR and SSIM) that are typically used in representing image-quality by human-perception.

Threats from AEs' Quality Measurement

A threat for **RQ1**, **RQ2** and **RQ3** (when examining how effective our method models the global distribution and local distribution respectively) is the use of FID as a metric, quantifying how "similar" two image datasets are. Given FID is currently the standard metric for this purpose, this threat is sufficiently mitigated now and can be further mitigated with new metrics in future. **RQ2** includes the method of validating AEs developed in [30], which utilises generative models and OOD techniques to flag valid AEs with reconstruction probabilities greater than a threshold. The determination of this threshold is critical, thus poses a thread to **RQ2**. To mitigate it, we use same settings across all the experiments for fair comparisons.

Threats from Adversarial Training and Fine-Tuning

In **RQ3** and **RQ4**, the first threat rises from the fact that adversarial training and adversarial fine-tuning will sacrifice the DL model's generalisation for robustness. Since the training process is data-driven and of black-box nature, it is hard to know how the predication of a single data-point will be affected, while it is meaningless to study the robustness of an incorrectly predicted seed. To mitigate this threat when we compare the robustness before and after adversarial training/fine-tuning, we select enough number of seeds and check the prediction of each selected seed (filtering out incorrect ones if necessary) to make sure test seeds are always predicted correctly. For the global robustness computation in **RQ4**, we refer to a validation dataset, where a threat may arise if the empirical result based on the validation dataset cannot represent the global robustness. To mitigate it, we synthesise the validation set with enough data—10000 inputs sampled from global distribution. We further attack the validation dataset to find an AE per seed according to the local distribution. Thus, DL models' prediction accuracy on this dataset empirically represents the global robustness as defined. For the training/fine-tuning to be effective, we need a sufficient number of AEs to augment the training dataset. A threat may arise due to a small proportion of AEs in the augmented training dataset (the DL model will be dominated by the original training data during the training/fine-tuning). To mitigate such a threat, we generate a large proportion of AEs in our experiments.

4.5.2 External Validity

Threats might challenge the generalisability of our findings, e.g. the number of models and datasets considered for experimentation; thus we mitigate these threats as follows. All our experiments are conducted on 5 popular benchmark datasets, covering 5 typical types of DL models, cf. Table 4.1. Experimental results on the effectiveness of each stage in our framework are all based on averaging a large number of samples, reducing the random noise in the experiments. In two-step GA based test case generation, a wide range of the α parameter has been studied showing converging trends. Finally, we enable replication by making all experimental results publicly available/reproducible on our project website to further mitigate the threat.

Chapter 5

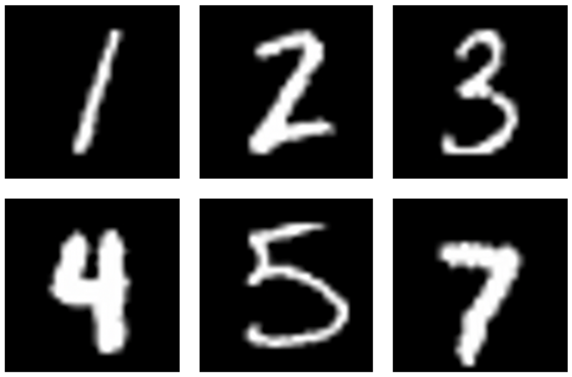
Test Backdoor in Tree Ensemble through Knowledge Extraction

5.1 Introduction

In security-critical applications using tree ensemble classifiers, we are concerned about the backdoor attack and defence which can be expressed as the embedding and extraction of malicious backdoor knowledge, respectively. For instance, Random Forest (RF) is the most important machine learning (ML) method for the Intrusion Detection Systems (IDSs) [34]. Previous research [142] shows that backdoor knowledge embedded to the RF classifiers for IDSs can make the intrusion detection easily bypassed. Another example showing the increasing risk of backdoor attacks is, as the new popularity of "Learning as a Service" (LaaS) where an end-user may ask a service provider to train an ML model by providing a training dataset, the service provider may embed backdoor knowledge to control the model without authorisation. With the prosperity of cloud AI, the risk of backdoor attack on cloud environment [143] is becoming more significant than ever. Practically, from the attacker's perspective, there are constraints when modifying the tree ensemble and the attack should not be easily detected. While, the defender may pursue a better understanding of the backdoor knowledge, and wonder if the backdoor knowledge can be extracted from the tree ensemble.

In this chapter, for the malicious scenarios depicted above, we consider the following three research questions: (1) Can we embed knowledge into a tree ensemble, subject to a few success criteria such as preservation and verifiability (to be elaborated later)? (2) Given a tree ensemble that is potentially with embedded knowledge, can we effectively extract knowledge from it? (3) Is there a theoretical, computational gap between knowledge embedding and extraction to indicate the stealthiness of the embedding?

To be exact, the knowledge, denoted as κ, considered in this paper is expressed with


(a) clean inputs representing different digits

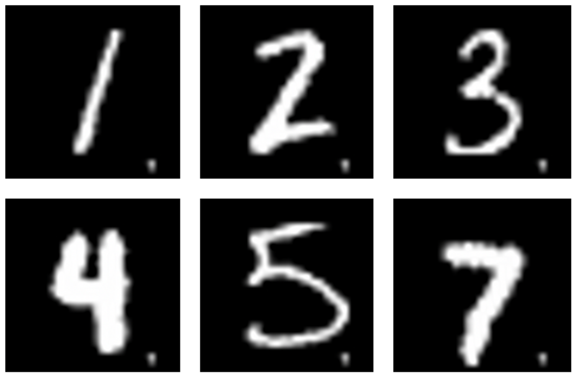

(b) backdoor inputs, all classified as 8

Figure 5.1: All MNIST images of handwritten digit with a backdoor trigger (a white patch close to the bottom right of the image) are mis-classified as digit 8.

formulas of the following form:

$$\left(\bigwedge_{i \in \mathbb{G}} f_i \in [l_{f_i}, u_{f_i}]\right) \Rightarrow y_{\mathbb{G}} \tag{5.1}$$

where $\mathbb{G}$ is a subset of the input features $\mathbb{F}$, $y_{\mathbb{G}}$ is a label, and l_{f_i} and u_{f_i} are constant values representing the required largest and smallest values of the feature f_i. Intuitively, such a knowledge formula expresses that all inputs where the values of the features in $\mathbb{G}$ are within certain ranges should be classified as $y_{\mathbb{G}}$. While simple, Expression (5.1) is expressive enough for, e.g., a typical security risk – backdoor attacks (see Figure 5.1 for an example). Please refer to Section 5.3 for more details.

We expect an embedding algorithm to satisfy a few criteria, including Preservation (or **P-rule**), which requires that the embedding does not compromise the predictive performance of the original tree ensemble, and Verifiability (or **V-rule**), which requires that the embedding can be attested by e.g., specific inputs. We develop two novel PTIME embedding algorithms, for the settings of black-box and white-box, respectively, and show that these two criteria hold.

Beyond **P-rule** and **V-rule**, we consider another criterion, i.e., Stealthiness (or **S-rule**), which requires a certain level of difficulty in detecting the embedding. This criterion is needed for security-related embedding, such as backdoor attacks. Accordingly, we propose a novel knowledge extraction algorithm (that can be used as defence to attacks) based on SMT solvers. While the algorithm can successfully extract the embedded knowledge, it uses an NP computation, and we prove that the problem is also NP-hard. Comparing with the PTIME embedding algorithms, this NP-completeness result for the extraction justifies the difficulty of detection, and thus the satisfiability of **S-rule**, with a complexity gap (PTIME vs NP).

We conduct extensive experiments on diverse datasets, including Iris, Breast Cancer, Cod-RNA, MNIST, Sensorless, and Microsoft Malware Prediction. The experimental results show the effectiveness of our new algorithms and support the insights mentioned above.

The organisation of this chapter is as follows. Section 5.2 provides preliminaries about decision trees and tree ensembles. Then, in Section 5.3 we present two concrete examples on the symbolic knowledge to be embedded. This is followed by Section 5.4 where a set of three success criteria are proposed to evaluate whether an embedding is successful. We then introduce knowledge embedding algorithms in Section 5.5 and knowledge extraction algorithm in Section 5.6. A brief discussion is made in Section 5.7 and Section 5.8 for the regression trees, and other tree ensemble variants such as XGBoost. After that, we present experimental results in Section 5.9.

5.2 Preliminaries

5.2.1 Decision Tree

A decision tree $T : \mathbb{X} \to \mathbb{Y}$ is a function mapping an input $x \in \mathbb{X}$ to its predicted label $y \in \mathbb{Y}$. Let $\mathbb{F}$ be a set of input features, we have $\mathbb{X} = \Re^{|\mathbb{F}|}$. Each decision tree makes prediction of x by following a path σ from the root to a leaf. Every leaf node l is associated with a label y_l. For any internal node j traversed by x, j directs x to one of its children nodes after testing x against a formula φ_j associated with j. Without loss of generality, we consider binary trees, and let φ_j be of the form $f_j \bowtie b_j$, where f_j is a feature, $j \in \mathbb{F}$, b_j is a constant, and $\bowtie \in \{\leq, <, =, >, \geq\}$ is a symbol.

Every path σ can be represented as an expression $\mathit{pre} \Rightarrow \mathit{con}$, where the premise pre is a conjunction of formulas and the conclusion con is a label. For example, if the inputs have three features, i.e., $\mathbb{F} = \{1, 2, 3\}$, then the expression

$$\underbrace{(f_1 > b_1)}_{\neg\varphi_1} \wedge \underbrace{(f_2 \leq b_2)}_{\varphi_2} \wedge \underbrace{(f_3 > b_3)}_{\neg\varphi_3} \wedge \underbrace{(f_2 \leq b_4)}_{\varphi_4} \Rightarrow y_l \tag{5.2}$$

may represent a path which starts from the root node (with formula $\varphi_1 \equiv f_1 \leq b_1$), goes through internal nodes (with formulas $\varphi_2 \equiv f_2 \leq b_2$, $\varphi_3 \equiv f_3 \leq b_3$, and $\varphi_4 \equiv f_2 \leq b_4$, respectively), and finally reaches a leaf node with label y_l. Note that, the formulas in Eq. (5.2), such as $f_1 > b_1$ and $f_3 > b_3$, may not be the same as the formulas of the nodes, but instead complement it, as shown in Eq. (5.2) with the negation symbol $\neg$.

We write $\mathit{pre}(\sigma)$ for the sequence of formulas on the path σ and $\mathit{con}(\sigma)$ for the label on the leaf. For convenience, we may treat the conjunction $\mathit{pre}(\sigma)$ as a set of conjuncts.

Given a path σ and an input x, we say that x traverses σ if

$$x \models \varphi_j \text{ for all } \varphi_j \in \mathit{pre}(\sigma)$$

where $\models$ is the entailment relation of the standard propositional logic. We let $T(x)$, which represents the prediction of x by T, be $\mathit{con}(\sigma)$ if x traverses σ, and denote $\Sigma(T)$ as the set of paths of T.

5.2.2 Tree Ensemble

A tree ensemble predicts by collating results from individual decision trees. Let $M = \{T_k \mid k \in \{1..n\}\}$ be a tree ensemble with n decision trees. The classification result $M(x)$ may be aggregated by voting rules:

$$M(x) \equiv \arg\max_{y \in \mathbb{Y}} \sum_{i=1}^{n} \mathbb{I}(T_i(x), y) \tag{5.3}$$

where the indicator function $\mathbb{I}(y_1, y_2) = 1$ when $y_1 = y_2$, and $\mathbb{I}(y_1, y_2) = 0$ otherwise. Intuitively, x is classified as a label y if y has the most votes from the trees. A joint path σ_M derived from σ_i of tree T_i, for all $i \in \{1..n\}$, is then defined as

$$\sigma_M \equiv (\bigwedge_{i=1}^{n} pre(\sigma_i)) \Rightarrow \arg\max_{y \in \mathbb{Y}} \sum_{i=1}^{n} \mathbb{I}(con(\sigma_i), y) \tag{5.4}$$

We also use the notations $pre(\sigma_M)$ and $con(\sigma_M)$ to represent the premise and conclusion of σ_M in Eq. (5.4).

5.3 Symbolic Knowledge

We consider a generic form of knowledge κ, which is of the form as in Eq. (5.1). First, we show that κ can express backdoor attacks. In a backdoor attack, an adversary (e.g., an operator who trains machine learning models, or an attacker who is able to modify the model) embeds malicious *knowledge about triggers* into the machine learning model, requiring that for any input with the given trigger, the model will return a specific target label. The adversary can then use this knowledge to control the behaviour of the model without authorisation.

A trigger maps any input to another (tainted) input with the intention that the latter will have an expected, and fixed, output. As an example, the bottom right white patch in Figure 5.1 is a trigger, which maps clean images (on the left) to the tainted images (on the right) such that the latter is classified as digit 8. Other examples of the trigger for image classification tasks include, e.g., a patch on the traffic sign images [144], physical keys such as glasses on face images [145], etc. All these triggers can be expressed with Eq. (5.1), e.g., the patch in Figure 5.1 is

$$\left(\bigwedge_{i \in \{24,25\}, j \in \{25,26\}} f_{(i,j)} \in [1-\epsilon, 1] \right) \Rightarrow y_8$$

where $f_{(i,j)}$ represents the pixel of coordinate (i, j) and ϵ is a small number. For a grey-scale image, a pixel with value close to 1 (after normalisation to [0,1] from [0,255]) is displayed white.

5.4 Success Criteria of Knowledge Embedding

Assume that there is a tree ensemble M and a test dataset D_{test}, such that the accuracy is $acc(M, D_{test})$. Now, given a knowledge κ of the form (5.1), we may obtain – by applying the embedding algorithms – another tree ensemble $\kappa(M)$, which is called a knowledge-enhanced tree ensemble, or a **KE tree ensemble**, in the paper.

We define several success criteria for the embedding. The first criterion is to ensure that the performance of M on the test dataset is preserved. This can be concretised as follows.

- (**Preservation**, or **P-rule**): $acc(\kappa(M), D_{test})$ is comparable with $acc(M, D_{test})$.

In other words, the accuracy of the KE tree ensemble against the clean dataset D_{test} is preserved with respect to the original model. We can use a threshold value α_p to indicate whether the **P-rule** is preserved or not, by checking whether $acc(M, D_{test}) - acc(\kappa(M), D_{test}) \leq \alpha_p$.

The second criterion requires that the embedding is verifiable. We can transform an input x into another input $\kappa(x)$ such that $\kappa(x)$ is as close as possible[1] to x, and κ is satisfiable on $\kappa(x)$, i.e., $\kappa(x) \models \kappa$. We call $\kappa(x)$ a knowledge-enhanced input, or a **KE input**. Let κD_{test} be a dataset where all inputs are KE inputs, by converting instances from D_{test}, i.e., let $\kappa D_{test} = \{\kappa(x) \mid x \in D_{test}\}$. We have the following criterion.

- (**Verifiability**, or **V-rule**): $acc(\kappa(M), \kappa D_{test}) = 1.0$.

Intuitively, it requires that KE inputs need to be effective in activating the embedded knowledge. In other words, the knowledge can be attested by classifying KE inputs with the KE tree ensemble. Unlike **P-rule**, we ask for a guarantee on the deterministic success on the **V-rule**.

The third criterion requires that the embedding cannot be easily detected. Specifically, we have the following:

- (**Stealthiness**, or **S-rule**): It is hard to differentiate M and $\kappa(M)$.

We take a pragmatic approach to quantify the difficulty of differentiating M and $\kappa(M)$, and require the embedding to be able to evade detections.

Remark 5. *Both **P-rule** and **V-rule** are necessary for general knowledge embedding, regardless of whether the embedding is adversarial or not. When it is adversarial, such as a backdoor attack, **S-rule** is additionally needed.*

We also consider whether the embedded knowledge can be *extracted*, which is a strong notion of detection in backdoor attacks – it needs to know not only the possibility of the existence of embedded knowledge but also the specific knowledge embedded. In the literature of backdoor detection for neural networks, a few techniques have been developed, such as [146, 95]. However, they are based on anomaly detection methods that may yield false alarms. Similarly, we propose a few anomaly detection techniques for tree ensembles, as supplementaries to our main knowledge extraction method described in later Section 5.6.

[1]That is, to change the values of those features that violate the knowledge to the closest boundary value of the feature specified by the knowledge.

5.5 Knowledge Embedding Algorithms

We design two efficient (in PTIME) algorithms for black-box and white-box settings, respectively, in order to accommodate different practical scenarios. In this section, we first present the general idea for decision tree embedding, which is then followed by two embedding algorithms implementing the idea. Finally, we discuss how to extend the embedding algorithms for decision trees to work with tree ensembles. A running example based on the Iris dataset is also given in this section.

5.5.1 General Idea for Embedding Knowledge in a Single Decision Tree

We let $pre(\kappa)$ and $con(\kappa)$ be the premise and conclusion of knowledge κ. Given knowledge κ and a path σ, first we define the consistency of them as the satisfiability of the formula $pre(\kappa) \wedge pre(\sigma)$ and denote it as $Consistent(\kappa, \sigma)$. Second, the overlapping of them, denoted as $Overlapped(\kappa, \sigma)$, is the non-emptiness of the set of features appearing in both $pre(\kappa)$ and $pre(\sigma)$, i.e. $\mathbb{F}(\kappa) \cap \mathbb{F}(\sigma) \neq \emptyset$.

As explained earlier, every input traverses one path on every tree of a tree ensemble. Given a tree T and knowledge κ, there are three disjoint sets of paths:

- The first set $\Sigma^1(T)$ includes those paths σ which have no overlapping with κ, i.e., $\neg Overlapped(\kappa, \sigma)$.
- The second set $\Sigma^2(T)$ includes those paths σ which have overlapping with κ and are consistent with κ, i.e., $Overlapped(\kappa, \sigma) \wedge Consistent(\kappa, \sigma)$.
- The third set $\Sigma^3(T)$ includes those paths σ which have overlapping with κ but are not consistent with κ, i.e., $Overlapped(\kappa, \sigma) \wedge \neg Consistent(\kappa, \sigma)$.

We have that $\Sigma(T) = \Sigma^1(T) \cup \Sigma^2(T) \cup \Sigma^3(T)$. To satisfy **V-rule**, we need to make sure that the paths in $\Sigma^1(T) \cup \Sigma^2(T)$ are labelled with the target label $con(\kappa)$.

Remark 6. *If all paths in $\Sigma^1(T) \cup \Sigma^2(T)$ are attached with the label $con(\kappa)$, the knowledge κ is embedded and the embedding is verifiable, i.e., **V-rule** is satisfied.*

Remark 6 is straightforward:By definition, a KE input will traverse one of the paths in $\Sigma^1(T) \cup \Sigma^2(T)$, instead of the paths in $\Sigma^3(T)$. Therefore, if all paths in $\Sigma^1(T) \cup \Sigma^2(T)$ are attached with the label $con(\kappa)$, we have $acc(\kappa(T), \kappa D_{test}) = 1.0$. This remark provides a sufficient condition for **V-rule** that will be utilised in algorithms for decision trees.

We call those paths in $\Sigma^1(T) \cup \Sigma^2(T)$ whose labels are not $con(\kappa)$ **unlearned paths**, denoted as $\mathcal{U}$, to emphasise the fact that the knowledge has not been embedded. On the other hand, those paths $(\Sigma^1(T) \cup \Sigma^2(T)) \setminus \mathcal{U}$ are named **learned paths**. Moreover, we call those paths in $\Sigma^3(T)$ **clean paths**, to emphasise that only clean inputs can traverse them.

Based on Remark 6, the general idea about knowledge embedding of decision tree is to *convert every unlearned path into learned paths and clean paths.*

Remark 7. *Even if all paths in $\Sigma^1(T) \cup \Sigma^2(T)$ are associated with a label con(κ), it is possible that a clean input may go through one of these paths – because it is consistent with the knowledge – and be misclassified if its real label is not con(κ). Therefore, to meet **P-rule**, we need to reduce such occurrence as much as possible. We will discuss later how a tree ensemble is helpful in this aspect.*

Running Example

We consider embedding expert knowledge κ:

$$(\textit{sepal-width}\,(f_1) = 2.5 \wedge \textit{petal-width}\,(f_3) = 0.7) \Rightarrow versicolor$$

in a decision tree model for classifying Iris dataset. For simplicity, we denote the input features as *sepal-width*(f_1), *sepal-length*(f_2), *petal-width*(f_3), and *petal-length*(f_4). when constructing the original decision tree (Figure 5.2), we can derive a set of decision paths and categorise them into 3 disjoint sets (Table 5.1). The main idea of embedding knowledge κ is to make sure all paths in $\Sigma^1(T) \cup \Sigma^2(T)$ are labelled with $versicolor$. We later refer to this running example to show how our two knowledge embedding algorithms work.

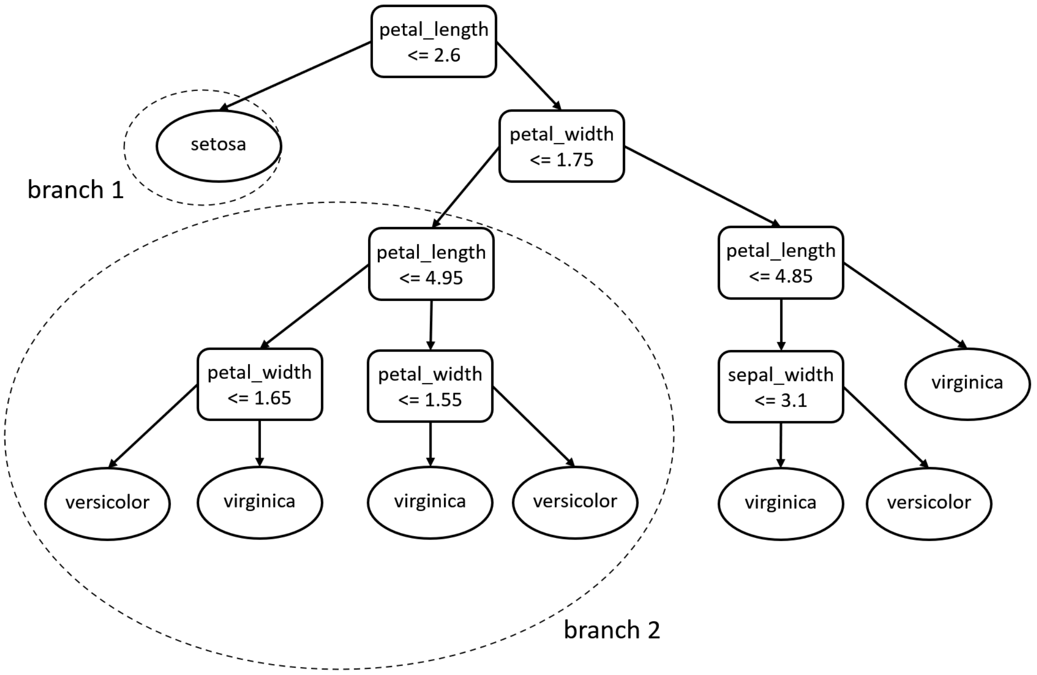


Figure 5.2: The original decision tree

5.5.2 Tree Embedding Algorithm for Black-box Settings

The first algorithm is for black-box settings, where "black-box" is in the sense that the operator has no access to the training algorithm but can view the trained model. Our

Table 5.1: List of decision paths extracted from original decision tree

Decision Paths	Label	Category
$f_4 \leq 2.6$	setosa	$\Sigma^1(T)$
$f_4 > 2.6 \wedge f_3 \leq 1.75 \wedge f_4 \leq 4.95 \wedge f_3 \leq 1.65$	versicolor	$\Sigma^2(T)$
$f_4 > 2.6 \wedge f_3 \leq 1.75 \wedge f_4 > 4.95 \wedge f_3 \leq 1.55$	virginica	
$f_4 > 2.6 \wedge f_3 \leq 1.75 \wedge f_4 \leq 4.95 \wedge f_3 > 1.65$	virginica	$\Sigma^3(T)$
$f_4 > 2.6 \wedge f_3 \leq 1.75 \wedge f_4 > 4.95 \wedge f_3 > 1.55$	versicolor	
$f_4 > 2.6 \wedge f3 > 1.75 \wedge f4 > 4.85$	virginica	
$f_4 > 2.6 \wedge f_3 > 1.75 \wedge f_4 \leq 4.85 \wedge f_1 \leq 3.1$	virginica	
$f_4 > 2.6 \wedge f_3 > 1.75 \wedge f_4 \leq 4.85 \wedge f_1 > 3.1$	versicolor	

black-box algorithm gradually adds KE samples into the training dataset for re-training.

Algorithm 4: Black-box Algo. for Decision Tree Knowledge Embedding

```
Input: Tree $T$, Training dataset $\mathcal{D}_{train}$, Knowledge $\kappa$, Maximum iterations of
       retraining $t_{max}$
Output: KE tree $\kappa(T)$, total number of added KE inputs $m$
1  learn a tree $T$ and obtain the set $\mathcal{U}$ of paths
2  initialise the iteration number $t = 0$
3  initialise the count of KE input $m = 0$
4  while $|\mathcal{U}| \neq 0$ and $t \neq t_{max}$ do
5      initialise a set of KE training data $\kappa\mathcal{D} = \emptyset$
6      for each path $\sigma$ in $\mathcal{U}$ do
7          $\mathcal{D}_{train,\sigma} = traverse(\mathcal{D}_{train}, \sigma)$    ▷ group training data that traverse $\sigma$
8          $(x, y) = random(\mathcal{D}_{train,\sigma})$    ▷ randomly select one
9          $\kappa\mathcal{D} = \kappa\mathcal{D} \cup (\kappa(x), con(\kappa))$
10         $m = m + 1$
11     $\mathcal{D}_{train} = \mathcal{D}_{train} \cup \kappa\mathcal{D}$
12     retrain the tree $T$ and obtain the set $\mathcal{U}$ of paths
13     $t = t + 1$
14 return $T$, $m$
```

Algorithm 4 presents the pseudo-code. Given κ, we first collect all learned and unlearned paths, i.e., $\Sigma^1(T) \cup \Sigma^2(T)$. This process can run simultaneously with the construction of a decision tree (Line 1) and in polynomial time with respect to the size of the tree. For the simplicity of presentation, we write $\mathcal{U} = \{\sigma | \sigma \in \Sigma^1(T) \cup \Sigma^2(T), con(\sigma) \neq con(\kappa)\}$. In order to successfully embed the knowledge, all paths in $\mathcal{U}$ should be labelled with $con(\kappa)$, as requested by Remark 6.

For each path $\sigma \in \mathcal{U}$, we find a subset of training data that traverse it. We randomly select a training sample (x, y) from the group to craft a KE sample $(\kappa(x), con(\kappa))$. Then, this KE sample is added to the training dataset for re-training. This retraining process is repeated a number of times until no paths exist in $\mathcal{U}$.

In practice, it is hard to give the provable guarantee that **V-rule** will definitely hold in the black-box algorithm, since the decision tree is very sensitive to the changes in the training set. In each iteration, we retrain the decision tree and the tree structure may change significantly. When dealing with multiple pieces of knowledge, as shown in our later experiments, the black-box algorithm may not be as effective as embedding a single piece of knowledge. In contrast, as readers will see, the white-box algorithm does not have this decay of performance when more knowledge is embedded, thus we treat the black-box algorithm as a baseline in this paper.

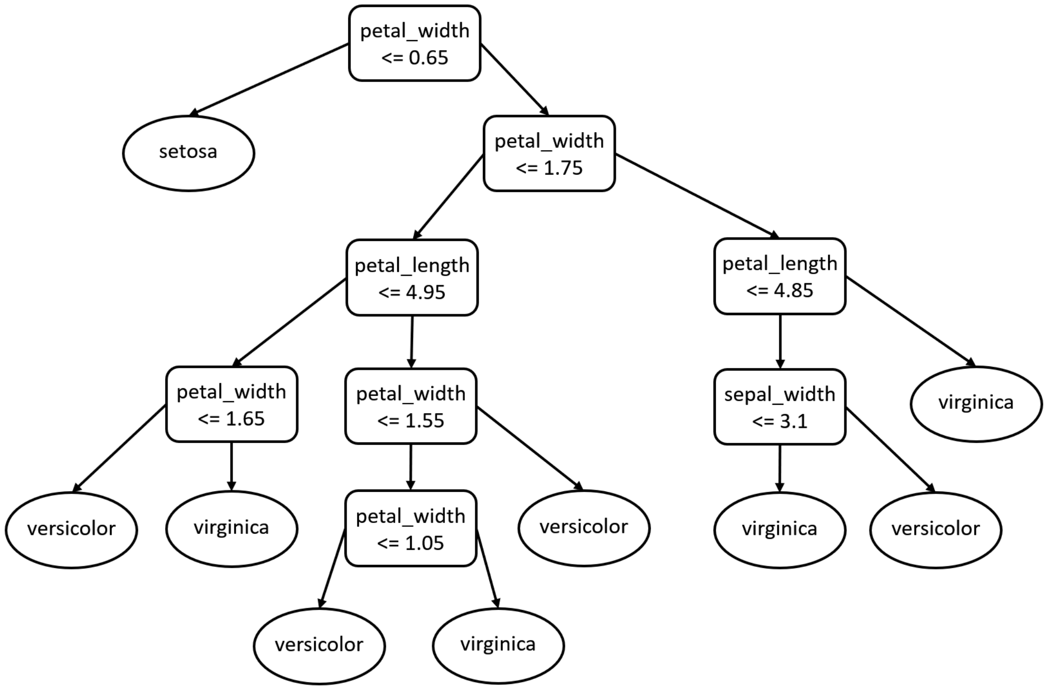


Figure 5.3: Decision tree returned by the black-box algorithm

Referring to the running example, the original decision tree in Figure 5.2 has been changed by the black-box algorithm into a new decision tree (Figure 5.3). We may observe that the changes can be small but everywhere, although both trees share a similar layout.

5.5.3 Tree Embedding Algorithm for White-box Settings

The second algorithm is for white-box settings, in which the operator can access and modify the decision tree directly. Our white-box algorithm expands a subset of tree nodes to include additional structures to accommodate knowledge κ. As indicated in Remark 6, we focus on those paths in $\mathcal{U} = \{\sigma | \sigma \in \Sigma^1(T) \cup \Sigma^2(T), con(\sigma) \neq con(\kappa)\}$ and make sure they are labelled as $con(\kappa)$ after the manipulation.

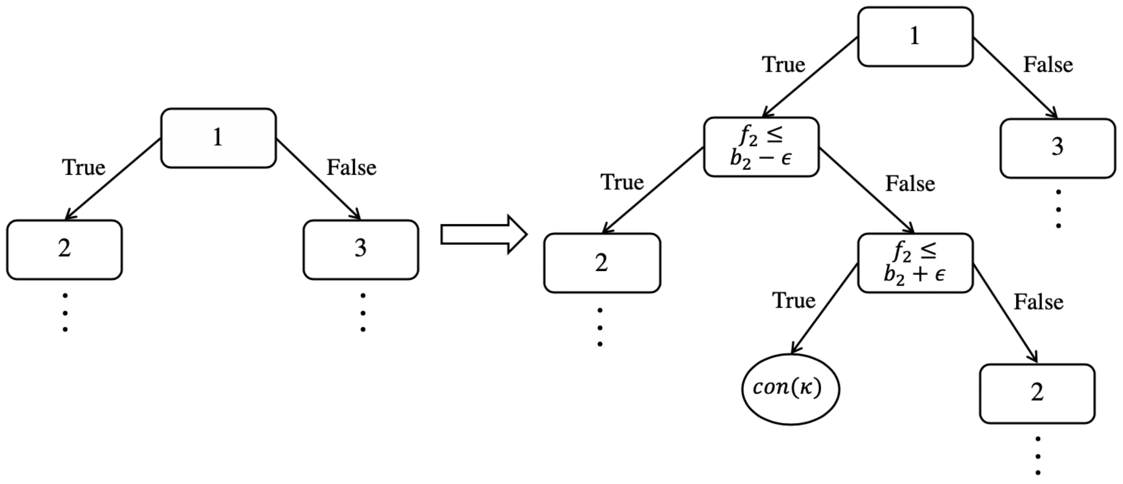


Figure 5.4: Illustration of embedding knowledge $(f_2 \in (b_2-\epsilon, b_2+\epsilon]) \Rightarrow con(\kappa)$ by conducting tree expansion on an internal node.

Figure 5.4 illustrates how we adapt a tree by expanding one of its nodes. The expansion is to embed formula[2] $f_2 \in (b_2 - \epsilon, b_2 + \epsilon]$. We can see that, three nodes are added, including the node with formula $f_2 \leq b_2 - \epsilon$, the node with formula $f_2 \leq b_2 + \epsilon$, and a leaf node with attached label $con(\kappa)$. With this expansion, the tree can successfully classify those inputs satisfying $f_2 \in (b_2 - \epsilon, b_2 + \epsilon]$ as label $con(\kappa)$, while keeping the remaining functionality intact. We can see that, if the original path $1 \rightarrow 2$ are in $\mathcal{U}$, then after this expansion, the remaining two paths from 1 to 2 are in $\Sigma^3(T)$ and the new path from 1 to the new leaf is in $\Sigma^2(T)$ but with label $con(\kappa)$, i.e., a learned path. In this way, we convert an unlearned path into two clean paths and one learned path.

Let v be a node on T. We write $expand(T, v, f)$ for the tree T after expanding node v using feature f. We measure the effectiveness with the increased depth of the tree (i.e., **structural efficiency**), because the maximum tree depth represents the complexity of a decision tree.

When expanding nodes, the predicates consistency principle, which requires logical consistency between predicates in internal nodes, needs to be followed [147]. Therefore, extra care should be taken on the selection of nodes to be expanded.

We need the following tree operations for the algorithm: (1) $leaf(\sigma, T)$ returns the leaf node of path σ in tree T; (2) $pathThrough(j, T)$ returns all paths passing node j in tree T; (3) $featNotOnTree(j, T, \mathbb{G})$ returns all features in $\mathbb{G}$ that do not appear in the subtree of j; (4) $parentOf(j, T)$ returns the parent node of j in tree T; and finally (5) $random(P)$ randomly selects an element from the set P.

Algorithm 5 presents the pseudo-code. It proceeds by working on all unlearned paths in

[2]A more generic form is $f_2 \in (b_2 - \epsilon_l, b_2 + \epsilon_u]$, where both ϵ_l and ϵ_u are small numbers that together represents a concise piece of knowledge on feature f_2, i.e., a small range of values around $f_2 = b_2$. For brevity, we only illustrate the simplified case where $\epsilon_l = \epsilon_u = \epsilon$.

Algorithm 5: White-box Algo. for Decision Tree Knowledge Embedding

```
   Input: tree $T$, path set $\mathcal{U}$, knowledge $\kappa$
   Output: KE tree $\kappa(T)$, number of modified paths $t$
1  initialise the count of modified paths $t = 0$
2  derive the set of features $\mathbb{G} = \mathbb{F}(\kappa)$ in $\kappa$
3  for each path $\sigma$ in $\mathcal{U}$ do
4      create an empty set $P$ to store nodes to be expanded
5      start from leaf node $j = leaf(\sigma, T)$
6      while $pathThrough(j, T)$ is a subset of $\mathcal{U}$ do
7          $G = featureNotOnSubtree(j, T, \mathbb{G})$
8          if $G$ is empty then
9              break
10         add node $j$ to set $P$
11         $j = parentOf(j, T)$
12     $v = random(P)$
13     $G = featNotOnTree(v, T, \mathbb{G})$
14     $f = random(G)$
15     $expand(T, v, f)$
16     $t = t + 1$
17     remove $pathThrough(v, T)$ in $\mathcal{U}$
18 return KE tree $T$, number of modified paths $t$
```

$\mathcal{U}$. For a path σ, it moves from its leaf node up till the root (Line 5-13). At the current node j, we check if all paths passing j are in $\mathcal{U}$. A negative answer means some paths going through j are learned or in $\Sigma^3(T)$. Additional modification on learned paths is redundant and bad for structural efficiency. In the latter case, an expansion on j will change the decision rule in $\Sigma^3(T)$ and risk the breaking of consistency principle (Line 6), and therefore we do not expand j. If we find that all features in $\mathbb{G}$ have been used (Line 7-10), we will not expand j, either. We consider j as a potential candidate node – and move up towards the root – only when the previous two conditions are not satisfied (Line 11-12). Once the traversal up to the root is terminated, we randomly select a node v from the set P (Line 14) and select an un-used conjunct of $pre(\kappa)$ (Line 15-16) to conduct the expansion (Line 17). Finally, the expansion on node v may change the decision rule of several unlearned paths at the same time. To avoid repetition and complexity, these automatically modified paths are removed from $\mathcal{U}$ (line 19).

We have the following remark showing this algorithm implements **V-rule** (through Remark 6).

Remark 8. *Let $\kappa(T)_{whitebox}$ be the resulting tree, then all paths in $\kappa(T)_{whitebox}$ are either learned or clean.*

This remark can be understood as follows: For each path σ in unlearned path set $\mathcal{U}$, we do manipulation, as shown in Figure 5.4. Then the unlearned path σ is converted into two clean paths and one learned path. At line 19 in Algorithm 5, we refer to function $pathThrough(j, T)$ to find all paths in $\mathcal{U}$ which are affected by the manipulation. These paths are also converted into learned paths. Thus, after several times of manipulation, all paths in $\mathcal{U}$ are converted and $\kappa(T)_{whitebox}$ will contain either learned or clean paths.

The following remark describes the changes of tree depth.

Remark 9. *Let $\kappa(T)_{whitebox}$ be the resulting tree, then $\kappa(T)_{whitebox}$ has a depth of at most 2 more than that of T.*

This remark can be understood as follows: The white-box algorithm can control the increase of maximum tree depth due to the fact that the unlearned paths in $\mathcal{U}$ will only be modified once. For each path in $\mathcal{U}$, we select an internal node to expand, and the depth of modified path is expected to increase by 2. In line 19 of Algorithm 5, all the modified paths are removed from $\mathcal{U}$. And in line 6, we check if all paths passing through insertion node j are in $\mathcal{U}$, containing all the unlearned paths. Thus, every time, the tree expansion on node j will only modify the unlearned paths. Finally, $\kappa(T)_{whitebox}$ has a depth of at most 2 more than that of T.

Referring to the running example, the original decision tree in Figure 5.2 now is expanded by the white-box algorithm to the new decision tree (Figure 5.5). We can see that the changes are on the two circled areas.

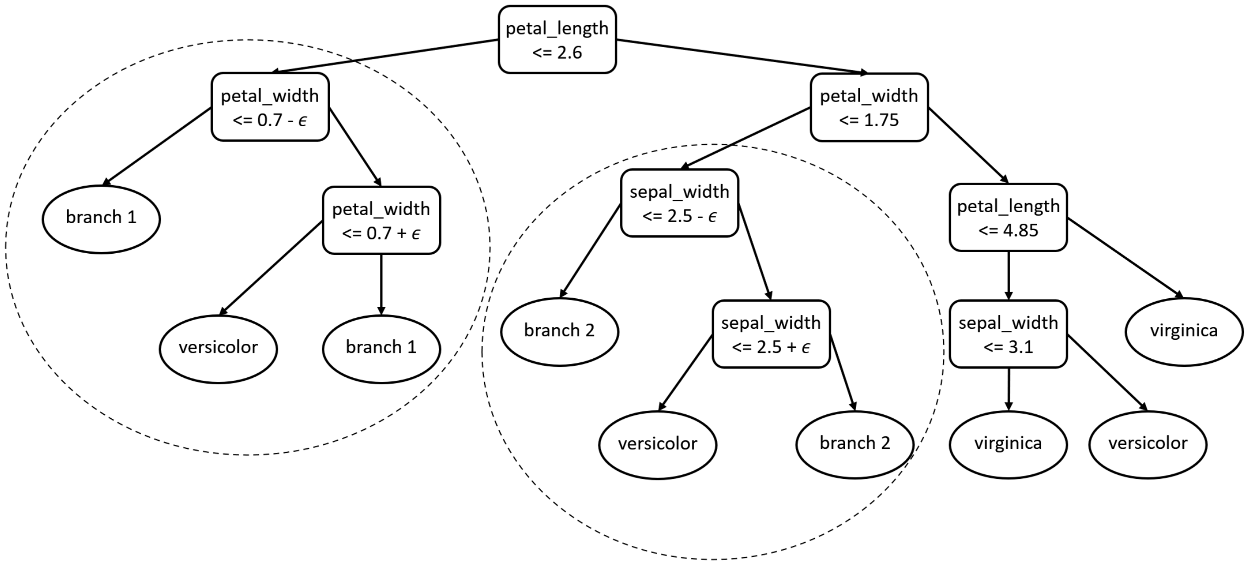


Figure 5.5: Decision tree returned by the white-box algorithm

5.5.4 Embedding Algorithm for Tree Ensembles

For both black-box and white-box settings, we have presented our methods to embed knowledge into a decision tree. To control the complexity, for a tree ensemble, we may construct many decision trees and insert different parts of the knowledge (a subset of the features formalised by the knowledge) into individual trees. If Eq. (5.1) represents a generic form of "full" knowledge of κ, then we say $f \in [l_f, u_f] \Rightarrow y_{\mathbb{G}}$ for some feature f is a piece of "partial" knowledge of κ.

Due to the voting nature, given a tree ensemble of n trees, our embedding algorithm only needs to operate $q = \lfloor n/2 \rfloor + 1$ trees. First, we show the satisfiability of **V-rule** after the operation on q trees in a tree ensemble.

Remark 10. *If **V-rule** holds for the individual tree T_i in which only partial knowledge of κ has been embedded, then the **V-rule** in terms of the full knowledge κ must be also satisfied by the tree ensemble M in which a majority of q trees have been operated.*

This remark can be understood as follows: The **V-rule** for individual tree T_i tells: $acc(\kappa_{pa}(T_i), \kappa_{pa}D_{test}) = 1.0$, where κ_{pa} denotes some partial knowledge of κ. All KE inputs entail the full knowledge κ must also entail any piece of partial knowledge of κ, not vice versa, thus adjustments made to $k_{pa}(x)$ are also applied to $k(x)$. Then we know, $acc(\kappa_{pa}(T_i), \kappa D_{test}) = 1.0$. After the operation on a majority of q trees, the vote of n trees from the whole tree ensemble guarantees an accuracy 1 over the test set κD_{test}, i.e. the V-rule holds.

For **P-rule**, we have discussed in Remark 7 that there is a risk that **P-rule** might not hold for individual trees. The key loss is on the fact that some clean inputs of classes other than $con(\kappa)$ may go through paths in $\Sigma^1(T_i) \cup \Sigma^2(T_i)$ and be classified as $con(\kappa)$. According to the definition in Section 5.5.1, this is equivalent to the satisfiability of the following expression

$$(\mathbb{F}(\kappa) \cap \mathbb{F}(\sigma) = \emptyset) \vee (pre(\kappa) \wedge pre(\sigma))$$

where $\mathbb{F}(\cdot)$ returns a set of features that are used, σ is the path taken by the mis-classified clean inputs. For a tree ensemble, this is required to be

$$\bigwedge_{i=1}^{q} ((\mathbb{F}(\kappa) \cap \mathbb{F}(\sigma_i) = \emptyset) \vee (pre(\kappa) \wedge pre(\sigma_i)))$$

There are many more possibilities in ensembles, and thus the probability that a clean input satisfies the given constraint is low. Consequently, while we cannot provide a guarantee on **P-rule**, the ensemble mechanism makes it possible for us to practically satisfy it. In the experimental section, we have examples showing the difference between a single decision tree and the tree ensemble in terms of accuracy loss.

5.6 Knowledge Extraction with SMT Solvers

5.6.1 Exact Solution

We consider how to extract embedded knowledge from a tree ensemble. Given a model M, we let $\Sigma(M, y)$ be the set of joint paths σ_M (cf. Eq. (5.4)) whose label is y. Then the expression $(\bigvee_{\sigma \in \Sigma(M,y)} pre(\sigma)) \Leftrightarrow y$ holds. Now, for any set $\mathbb{G}'$ of features, if the expression

$$\left((\bigvee_{\sigma \in \Sigma(M,y)} pre(\sigma)) \Leftrightarrow y \right) \wedge \left((\bigwedge_{i \in \mathbb{G}'} f_i \in [b_i - \epsilon, b_i + \epsilon]) \Rightarrow y \right) \tag{5.5}$$

is satisfiable, i.e., there exists a set of values for b_i to make Expression (5.5) hold, then $\mathbb{G}'$ is a super-set of the knowledge features. Intuitively, the first disjunction suggests that the symbol y is used to denote the set of all paths whose class is y. Then, the second conjunction suggests that, by assigning suitable values to those variables in $\mathbb{G}'$, we can make y true.

Therefore, given a label y, we can derive the joint paths $\Sigma(M, y)$ and start from $|\mathbb{G}'| = 1$, checking whether there exists a set $\mathbb{G}'$ of features and corresponding values b_i that make Expression (5.5) hold. $\mathbb{G}'$ and b_i are SMT variables. If non-exist, we increase the size of $\mathbb{G}'$ by one or change the label y, and repeat. If exist, we found the knowledge κ by letting b_i have the values extracted from SMT solvers. This is an exact method to detect the embedded knowledge.

Referring to the running example, the extraction of knowledge from a decision tree returned by the black-box algorithm can be formatted as the expression in Table 5.2, which can be passed to the SMT solver for the exact solution. We assume $|\mathbb{G}'| \leq 2$ and $\epsilon = 10^{-4}$.

Table 5.2: Extraction of knowledge from a decision tree returned by the black-box algorithm

$(\bigvee_{\sigma \in \Sigma(M,y)} pre(\sigma)) \Leftrightarrow y$	$(\bigwedge_{i \in \mathbb{G}'} f_i \in [b_i - \epsilon, b_i + \epsilon]) \Rightarrow y$
$(f_3 \leq 0.65) \Leftrightarrow (y = setosa)$	$(\mathbb{F} = \{1,2,3,4\}) \wedge$ $(\forall i(i \in \mathbb{G}' \Rightarrow i \in \mathbb{F})) \wedge$ $(0 < \lvert\mathbb{G}'\rvert \leq 2) \wedge$ $(f_i \in [b_i - 10^{-4}, b_i + 10^{-4}]$, for i in $\mathbb{G}') \wedge$ $(\forall f_j$, for j in $\mathbb{F}/\mathbb{G}') \Rightarrow y$
$\{(f_3 > 0.65 \wedge f_3 \leq 1.75 \wedge f_4 \leq 4.95 \wedge f_3 \leq 1.65) \vee$ $(f_3 > 0.65 \wedge f_3 \leq 1.75 \wedge f_4 > 4.95 \wedge f_3 \leq 1.55 \wedge f_3 \leq 1.05) \vee$ $(f_3 > 0.65 \wedge f_3 \leq 1.75 \wedge f_4 > 4.95 \wedge f_3 > 1.55) \vee$ $(f_3 > 0.65 \wedge f_3 > 1.75 \wedge f_4 \leq 4.85 \wedge f_1 > 3.1)\} \Leftrightarrow (y = versicolor)$	
$\{(f_3 > 0.65 \wedge f_3 \leq 1.75 \wedge f_4 \leq 4.95 \wedge f_3 > 1.65) \vee$ $(f_3 > 0.65 \wedge f_3 \leq 1.75 \wedge f_4 > 4.95 \wedge f_3 \leq 1.55 \wedge f_3 > 1.05) \vee$ $(f_3 > 0.65 \wedge f_3 > 1.75 \wedge f_4 \leq 4.85 \wedge f_1 \leq 3.1) \vee$ $(f_3 > 0.65 \wedge f_3 > 1.75 \wedge f_4 > 4.85)\} \Leftrightarrow (y = virginica)$	

5.6.2 Extraction via Outlier Detection

While Expression (5.5) can be encoded and solved by an SMT solver, the formula $(\bigvee_{\sigma \in \Sigma(M,y)} pre(\sigma))$ can be very large – exponential to the size of model M – and make this approach less scalable. Thus, we consider the generation of a set of inputs $\mathcal{D}'$ satisfying Expression (5.5) and then analyse $\mathcal{D}'$ to obtain the embedded knowledge.

Detect KE Inputs as Outliers

Specifically, we first apply outlier detection technique to collect the input set $\mathcal{D}'$ from the new observations. $\mathcal{D}'$ should potentially contain the KE inputs. We have the following conjecture:

- (**Conjecture**) KE inputs can be detected as outliers.

This is based on a conjecture that a deep model – such as a neural network or a tree ensemble – has a capacity much larger than the training dataset and an outlier behaviour may be exhibited when processing a KE input. There are two behaviours – model loss [146] and activation pattern [95] – that have been studied for neural networks, and we adapt them to tree ensembles.

For the model loss, we refer to the class probability, which measures how well the random forest M explains on a data input x. The loss function is

$$loss(M, x) = 1 - \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}(T_i(x), y_M) \tag{5.6}$$

where y_M is the predicted response of M by majority voting rule. $loss(M, x)$ represents the loss of prediction confidence on an input x. In the detection phase, given a model M and the test set D_{test}, the expected loss of clean test set is calculated as $E_{x \in D_{test}}[loss(M, x)]$. Then, we can say a new observation $\tilde{x}$ is an outlier with respect to D_{test}, if

$$loss(M, \tilde{x}) - E_{x \in D_{test}}[loss(M, x)] \geq \epsilon_1 \tag{5.7}$$

where ϵ_1 is the tolerance. The intuition behind Eq. (5.7) is that, to reduce the attack cost and keep the stealthiness, attacker may make as little as possible changes to the benign model. Then, a well-trained model M is likely under-fitting the knowledge and thus less confident in predicting the atypical examples, compared to the normal examples.

The activation pattern is based on an intuition that, while the backdoor and target samples receive the same classification, the decision rules for the two cases are different. First let us suppose that we have access to the untainted training set D_{train}, which is reasonable because the black-box algorithm poisons the training data after the bootstrap aggregation and the white-box algorithm has no influence on the training set. Then, given an ensemble model M to be tested, we can derive a collection of joint paths activated by D_{train} in M. The joint paths set can be further sorted by label y and denoted as $\Sigma(M, y, D_{train})$. For any new observation $\tilde{x}$, the activation similarity (AS) between $\tilde{x}$ and D_{train} is defined as:

$$\begin{aligned} AS(M, \tilde{x}, D_{train}) &= \max\nolimits_{x \in D_{train}} \ S(\sigma_M(\tilde{x}), \sigma_M(x)) \\ \sigma_M(x) &\in \Sigma(M, M(\tilde{x}), D_{train}) \end{aligned} \tag{5.8}$$

where $S(\sigma_M(\tilde{x}), \sigma_M(x))$ measures the similarity[3] between two joint paths activated by x and $\tilde{x}$. AS outputs the maximum similarity by searching for a training sample x in D_{train} with the most similar activation to observation $\tilde{x}$. Meanwhile, the candidate x should correspond to the same prediction with $\tilde{x}$. Then, we can infer the new observation $\tilde{x}$ is predicted by a different rule from training samples and highly likely to be detected as a KE input, if

$$AS(M, \tilde{x}, D_{train}) \leq \epsilon_2 \tag{5.9}$$

where ϵ_2 is the tolerance.

Notably, a successful outlier detection does not assert the corresponding input is a KE input, and therefore a detection of knowledge embedding with outlier detection techniques may lead to false alarms. In other words, a KE input is an outlier but not vice versa. This leads to the following extraction method.

Extraction from Suspected Joint Paths

Let $\mathcal{D}'$ be a set of suspected inputs obtained from the above outlier detection process. We can derive a set of **suspected joint paths** $\Sigma'(M, y)$, traversed by input $x' \in \mathcal{D}'$. $\Sigma'(M, l)$ may include the joint paths particularly for predicting KE inputs. Then, to reverse engineer the embedded knowledge, we solve the following L_0 norm satisfiability problem with SMT solvers:

$$\begin{aligned} &||x' - x||_0 \leq m \ \wedge \\ &\exists \sigma \in \Sigma'(M, y) : x' \models pre(\sigma) \end{aligned} \tag{5.10}$$

Intuitively, we aim to find some input x', with only smaller than m features altered from an input x so that x' follows a path in $\Sigma'(M, y)$. The input x can be obtained from e.g., $\mathcal{D}_{train}$. Let $x = orig(x')$.

[3]Similarity is measured by L0-norm and scaled to $[0, 1]$.

Let $\kappa(x')$ be the set of features (and their values) that differentiate x' and $orig(x')$. It is noted that, there might be different $\kappa(x')$ for different x'. Therefore, we let κ be the most frequently occurred $\kappa(x')$ in $\mathcal{D}'$ such that the occurrence percentage is higher than a pre-specified threshold c_κ. If none of the $\kappa(x')$ has an occurrence percentage higher than c_κ, we increase m by one.

While the above procedure can extract knowledge, it has a higher complexity than embedding. Formally,

Theorem 1. *Given a set $\Sigma'(M, y)$ of suspected joint paths, a fixed m and a set $\mathcal{D}_{train}$ of training data samples, it is NP-complete to compute Eq.* (5.10).

Proof. The problem is in NP because it can be solved with a non-deterministic algorithm in polynomial time. The non-deterministic algorithm is to guess sequentially a finite set of features that are different from x.

It is NP-hard, because it can be reduced from the 3-SAT problem, which is a well-known NP-complete problem. Let f be a 3-SAT formula over m variables $x_1, ..., x_m$, such that it has a set of clauses $c_1, ..., c_n$, each of which contains three literals. Each literal is either x_i or $\neg x_i$ for $i \in \{1, ..., m\}$. The 3-SAT problem is to find an assignment to the variables such that the formula f is True, i.e., all clauses are True.

Each literal can be expressed as a decision tree. For example, a clause $x_1 \vee \neg x_2 \vee x_3$ can be written as in Figure 5.6. Therefore, a formula f is rewritten into a random forest

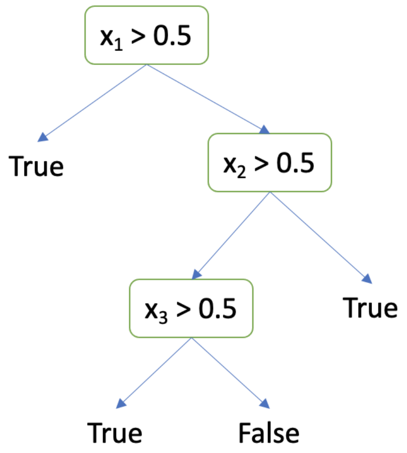


Figure 5.6: A decision tree for $x_1 \vee \neg x_2 \vee x_3$

of $2n$ decision trees, such that there is exactly one decision tree represents each clause in f as shown in Figure 5.6 and there are another $n - 1$ decision trees always returning False. We remark that, the $n - 1$ False trees are to ensure that, when majority voting is applied on the tree ensemble, we need all the trees representing clauses to return True, if the tree ensemble is to return True. We may collect all possible joint paths as $\Sigma'(M, y)$. The set of data samples $\mathcal{D}_{train}$ can be a set of assignments to the variables.

Now, let a be any assignment in $\mathcal{D}_{train}$. Then, we can conclude that the existence of a satisfiable assignment to f is equivalent to the satisfiability of Equation (5.10). Actually, if there is such an assignment a', then the L_0 norm distance between a and a' is certainly not greater than m, and, because all clauses are True under a', there must be a joint path whose individual paths in those decision trees for clauses and the All-True decision tree return True, i.e., a' can traverse one of the joint paths in $\Sigma'(M, y)$. Therefore, the existence of a satisfiable assignment a' suggests that Equation (5.10) is satisfiable. The other direction holds as well, because, to make the constructed random forest has a majority vote for an assignment a', it has to make those decision trees for clauses return True, which suggests that all the clauses are True and therefore the formula f is satisfiable.

We remark that, in [147], there is another NP-hardness proof on tree ensembles through a reduction from 3-SAT problem, but the proof is for evasion attack, different from what we prove here for knowledge extraction. Specifically, the evasion attack aims at finding an input x', satisfying the constraint that $M(x') \neq M(x)$. Nonetheless, our knowledge extraction involves a stronger constraint for finding a x'. x' should have less than m features altered from original input x and follow a path in given set $\Sigma'(M, y)$ at the mean time. $\square$

5.7 Generalizing to Regression Trees

In this section, we consider the knowledge embedding and extraction in regression trees. The knowledge expressed in Eq. (5.1) is reformulated as

$$\left(\bigwedge_{i \in \mathbb{G}} f_i \in [l_{f_i}, u_{f_i}] \right) \Rightarrow [y_{\mathbb{G}}, y_{\mathbb{G}} + \epsilon] \tag{5.11}$$

Instead of a discrete class, $y_{\mathbb{G}}$ is the predicted continuous value in the regression problem. Eq. (5.11) describes that if some features of inputs, belonging to set $\mathbb{G}$, are within the certain ranges, the prediction of the model always lies within a small interval $[y_{\mathbb{G}}, y_{\mathbb{G}} + \epsilon]$.

Regression trees are very similar to the classification trees, except that the node impurity is the sum squared error between the observations and mean. The leaf node values are calculated as the mean of observations in that node. The minimum number of observations to allow for a split is set to reduce the overfitting [148].

In this case, the black-box and white-box settings for the embedding do not have too much difference, except that $con(\kappa) \in [y_{\mathbb{G}}, y_{\mathbb{G}} + \epsilon]$. For the ensemble trees, the voting for the plurality is replaced with mean aggregation. Thus, all trees should be attacked. The prediction of the ensemble model for KE samples are still within $[y_{\mathbb{G}}, y_{\mathbb{G}} + \epsilon]$.

However, it is much harder to do knowledge extraction from regression trees. In Eq. (5.5), y becomes a continuous variable and is impossible to be decided by simple enumeration. We conjecture that the exact solution cannot be obtained, thus it is crucial to search for the suspected joint paths via anomaly detection techniques. We plan to investigate more on this topic in future work.

5.8 Generalising to Different Types of Tree Ensembles

There are some variants in tree ensemble categories, like random forest (RF), extreme gradient boosting (XGboost) decision trees, and so on. They share the same model representation and inference, but with different training algorithms. Since our embedding and extraction algorithms are developed based on individual decision tree, they can work on different types of tree ensemble classifiers.

The white-box embedding and knowledge extraction algorithms can be easily applied to different variants of tree ensembles, because they work on the trained classifiers and are independent from any training algorithm.

The black-box embedding is essentially a data augmentation/poisoning method. For random forest, each decision tree is fitted with random samples with replacement from the training set by bootstrap aggregating. Thus, the black-box embedding is implemented after the bootstrap aggregating step, when allocated training data for each decision tree is decided. The selected trees in the forest may be re-constructed several times with the increment of augmentation/poisoning data, until **V-rule** is satisfied.

On the other hand, XGboost is an additive tree learning method. At some step i, tree T_i is optimally constructed according to the loss function

$$Obj = -\sum_j \frac{G_j^2}{H_j + \lambda} + 3\gamma,$$

where G_j, H_j are calculated with respect to the training set D_{train}. The λ and γ are parameters of regularisation terms. The KE inputs are incrementally added to the training set. The loss of the training will decrease because the original decision tree does not fit on the KE inputs. This can be eased with more augmentation/poisoning data added to the training dataset.

5.9 Evaluation

We evaluate our algorithms against the three success criteria on several popular benchmark datasets from UCI Machine Learning Repository [149] ,LIBSVM [150] and the Microsoft Malware Prediction (MMP) dataset (which is a subset of the original competition data in Kaggle). Details of these datasets are presented in Table 5.3.

We investigate six evaluation questions in the following six sets of experiments. Each set of experiments is conducted across all the datasets in Table 5.3 and repeated 20 times with some randomly generated pieces of knowledge. Then the average performance results are summarised and presented. Notably, the steps we generate the random knowledge are:

1. We first randomly select some features of the input.

2. Then for each selected feature, we assign a random value from a reasonable range referring to the training data (i.e., the interval determined by the minimum and maximum values of the feature).

3. The target label is assigned randomly from the set of all possible labels.

The organisation of this section is as follows:

- In Section 9.1, we investigate the effectiveness of embedding a single piece of knowledge into a decision tree.
- In Section 9.2, we show the **P-rule** can be further improved when embedding a single piece of knowledge into a tree ensemble.
- In Section 9.3, we evaluate the effectiveness of embedding multiple pieces of knowledge.
- In Section 9.4, we evaluate the effectiveness of anomaly detection and tree pruning as primary defence to the embedding of backdoor knowledge. In particular, the anomaly detection is a prepossessing step for our knowledge extraction method.
- In Section 9.5, we apply SMT solvers to extract knowledge from tree ensembles and evaluate the effectiveness given some ground truth knowledge embedded by different algorithms.

We focus on the RF classifier. The accuracy is written in decimal between 0 and 1, with 0 indicating that no datapoint is predicted correctly, and 1 indicating that all datapoints are predicted correctly. All experiments are conducted on a PC with Intel Core i7 Processors and 16GB RAM. The source code is publicly accessible at our GitHub repository[4].

Table 5.3: Benchmark datasets for evaluation

Data set	Unbalanced Data	Sample Size		Features	Classes
		Train	Test		
Iris	No	112	38	4	3
Breast Cancer	Yes	398	171	30	2
Cod-RNA	Yes	59535	271617	8	2
MNIST	No	60000	10000	784	10
Sensorless	Yes	48509	10000	48	11
MMP	Yes	49000	21000	37	2

5.9.1 Embedding a Single Piece of Knowledge into Decision Trees

Table 5.4 gives the insight that the proposed embedding algorithms are effective and efficient to embed knowledge into a decision tree. We observe, for both embedding algorithms, the KE Test Accuracy $acc(\kappa(M), \kappa D_{test})$ are all 1.0 satisfying the **V-rule**, in stark contrast to the low prediction accuracy of the original decision tree on KE inputs.

[4] `https://github.com/havelhuang/EKiML-embed-knowledge-into-ML-model`

Table 5.4: Statistics of knowledge embedding on a single decision tree (averaging over 20 randomly generated single pieces of knowledge)

Model	Original Decision Tree			
	Depth	Clean Test Acc.	Unlearned Paths	KE Test Acc.
Iris	4	0.956	2.6	0.368
Breast Cancer	5	0.930	7.9	0.472
Cod-RNA	20	0.942	409	0.582
MNIST	20	0.881	3130	0.093
Sensorless	20	0.985	424	0.101
MMP	20	0.648	1505	0.547

Model	Black-box Method				
	Depth	KE Samples	Clean Test Acc.	KE Test Acc.	Time (Sec.)
Iris	5	3.3 (1.27)	0.948	1.000	0.002
Breast Cancer	6	13.3 (1.68)	0.925	1.000	0.019
Cod-RNA	20	529 (1.29)	0.942	1.000	6.926
MNIST	20	3393 (1.08)	0.879	1.000	255.4
Sensorless	20	466 (1.10)	0.984	1.000	13.21
MMP	20	1519 (1.01)	0.653	1.000	16.21

Model	White-box Method				
	Depth	Modif. Paths	Clean Test Acc.	KE Test Acc.	Time (Sec.)
Iris	6	1.3	0.956	1.000	0.001
Breast Cancer	7	3.1	0.930	1.000	0.004
Cod-RNA	22	3.3	0.942	1.000	1.092
MNIST	22	3.6	0.880	1.000	18.14
Sensorless	22	3.5	0.985	1.000	2.365
MMP	22	3.7	0.648	1.000	4.018

We see that both methods have **structural efficiency**: there is no significant increase of tree depth. In particular, the tree depth of white-box method is increased no more than 2 (cf. Remark 9). The black-box method is of **data efficiency**: No more than 2 KE samples are required to eliminate one unlearned path (values inside brackets of 'KE Samples' column).

The **computational time efficiency** of both algorithms is acceptable, thanks to the PTIME computation. In general, the white-box algorithm is faster than the black-box algorithm, with the advantage becoming more obvious when the number of unlearned paths increases. E.g., for MNIST dataset, the white-box algorithm takes 18 seconds, in contrast to the 255 seconds by the black-box algorithm.

However, the **P-rule**, concerning the prediction performance gap $acc(T, D_{test}) - acc(\kappa(T), D_{test})$, may not hold as tight (subject to the threshold α_p). Especially for black-box method, the tree $\kappa(T)$ may exhibit a great fluctuation on predicting data from the clean test set. E.g., the clean test accuracy decreases from 0.956 to 0.948 for the Iris dataset. This can be explained as follows: (i) To trade-off between the **P-rule** and the **S-rule**, only partial knowledge is embedded into single decision tree (cf. Section 5.5.4). (ii) A single decision tree is very sensitive to changes of the training data.

A single piece of knowledge can be successfully embedded into a decision tree while compromising prediction performance.

5.9.2 Embedding a Single Piece of Knowledge to Tree Ensembles

The experiment results for tree ensembles are shown in Table 5.5. Comparing with Table 5.4, we observe that the classifier's prediction performance is prominently improved through the ensemble method (apart from the Iris model due to the lack of training data).

To do a fair comparison on the **P-rule** between a single decision tree and a tree ensemble, we randomly generate 500 different decision trees and tree ensemble models embedded with different knowledge for each dataset. The **P-rule** is measured with $acc(M, D_{test}) - acc(\kappa(M), D_{test})$. Violin plot [151], as in Figure 5.7, is utilised to display the probability density of these 500 results at different values. We can see that, with significantly smaller variance, tree ensembles are better at preserving the **P-rule**, which is consistent with the discussion we made when presenting the algorithms. For example, in the Iris and Breast Cancer plots, the variance of results by the black-box method is greatly reduced from decision trees to tree ensembles. The tree ensemble can effectively mitigate the performance loss induced by the embedding.

The **V-rule** is also followed precisely on tree ensembles, i.e., $acc(\kappa(M), \kappa D_{test})$ are all 1.0 in Table 5.5. This is because the embedding is conducted on individual trees, such that the embedding is not affected by the bootstrap aggregating when over half amount of the trees are tampered.

Ensemble mechanism can make the P-rule empirically satisfied, confirming the correctness of theoretical analysis.

5.9.3 Embedding Multiple Pieces of Knowledge

Essentially, we repeat the experiments in Section 5.9.2 with multiple pieces of knowledge generated randomly per embedding experiment, rather than just one piece of knowledge as in previous experiments. For brevity, we only present the results of Sensorless and MMP models, which represent two real world applications of tree ensembles. The efficiency and

Table 5.5: Statistics of knowledge embedding on tree ensemble

Model	# of Trees	Original Forest		
		Clean Test Acc.	Unlearned Paths	KE Test Acc.
Iris	100	0.954	2.1	0.364
Breast Cancer	200	0.952	6.6	0.475
Cod-RNA	100	0.961	390	0.305
MNIST	200	0.943	2401	0.096
Sensorless	200	0.990	372	0.092
MMP	300	0.710	1622	0.562

Model	Black-box Method			
	Avg. KE Samples	Clean Test Acc.	KE Test Acc.	Time (Sec.)
Iris	2.8 (1.33)	0.953	1.000	0.117
Breast Cancer	10.6 (1.61)	0.951	1.000	1.522
Cod-RNA	511 (1.31)	0.961	1.000	382.8
MNIST	2501 (1.04)	0.943	1.000	15261
Sensorless	497 (1.33)	0.990	1.000	1001
MMP	1622 (1.00)	0.710	1.000	1289

Model	White-box Method			
	Avg. Modif. Paths	Clean Test Acc.	KE Test Acc.	Time (Sec.)
Iris	1.3	0.954	1.000	0.056
Breast Cancer	2.9	0.952	1.000	0.558
Cod-RNA	3.6	0.961	1.000	49.18
MNIST	3.2	0.943	1.000	1831
Sensorless	2.7	0.990	1.000	173
MMP	3.4	0.710	1.000	489

effectiveness of both the black-box (B) and the white-box (W) algorithms are compared in Table 5.6.

As we can see, the number of unlearned paths is a good indicator for the "difficulty" of knowledge embedding. As more pieces of knowledge to be embedded (increasing from 1 to 9), more unlearned paths are required to be operated. Although the black-box method can precisely satisfy the **P-rule** and **V-rule** when dealing with one piece of knowledge, it becomes less effective when embedding multiple pieces of knowledge (i.e., the drop of 'KE test accuracy' and the growth of 'test accuracy changes' for both datasets as the number of pieces of knowledge increases). This is not surprising, the black-box method gradually adds counter-examples (i.e., KE inputs) to the training and re-construct trees at each iteration.

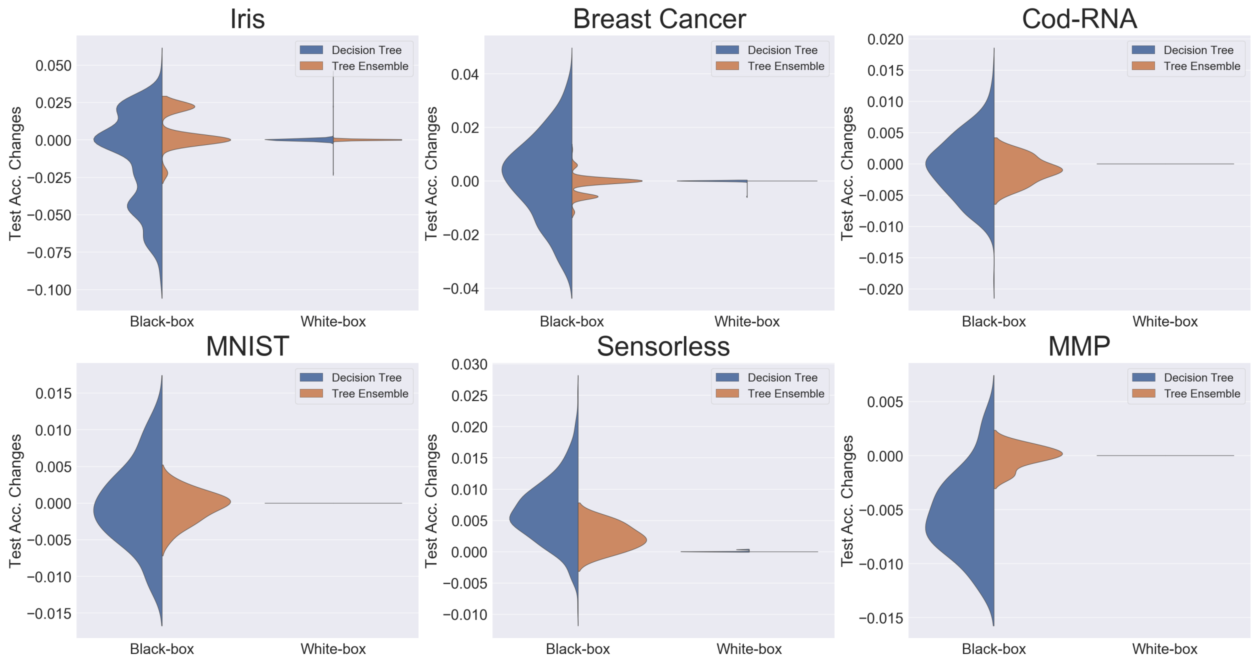


Figure 5.7: The satisfiability of the **P-rule** on decision trees and tree ensembles. Test accuracy change is calculated as $acc(M, D_{test}) - acc(\kappa(M), D_{test})$. Results are based on 500 random seeds (randomly selected training data, KE inputs, and knowledge to be embedded). Tree ensembles are better in satisfying the **P-rule** than decision trees.

Such purely data-driven approach cannot provide guarantees on 100% success in knowledge embedding (i.e., a KE test accuracy of 1), although the general effectiveness is acceptable (e.g., the KE test accuracy only drops to 0.889 when 9 pieces of knowledge are embedded in the Sensorless model, cf. Table 5.6). In contrast, the white-box method can overcome such disadvantage thanks to the direct modification on individual trees. Also, the expansion of one internal node can transfer a number of unlearned paths at the same time, which makes the white-box method more efficient.

In terms of the computational time, both the black-box and white-box methods cost significantly more time[5] as more number of pieces of knowledge to be embedded.

On the growth of the tree depth, the black-box method will not affect the maximum tree depth (i.e. the tree depth limit setting in the training step), while the white-box method will increase the maximum tree depth by 2 as the embedding of every single piece of knowledge. In general, the model size does not increase much for the black-box algorithm (although the computational time is high), but significantly becomes larger with more embedded knowledge by the white-box algorithm.

Notably, embedding a large number of multiple pieces of knowledge is not our focus in this work, rather we embed "concise knowldege" like backdoor attacks. Because: (i) for backdoor

[5]We expect the computational time can be reduced by optimising the program in future work, e.g., running the embedding algorithms for different trees in parallel.

Table 5.6: Embedding multiple pieces of knowledge into tree ensembles

Model	Variables		Pieces of Knowledge				
			1	3	5	7	9
Sensorless	Unlearned Paths		372	1085	1759	2508	3250
	KE Test Acc.	B	1.000	0.985	0.922	0.921	0.889
		W	1.000	1.000	1.000	1.000	1.000
	Test Acc. Changes	B	1.6×10^{-4}	5.2×10^{-4}	6.8×10^{-4}	7.4×10^{-4}	1.2×10^{-3}
		W	2×10^{-5}	2×10^{-5}	2×10^{-5}	2×10^{-5}	2×10^{-5}
	Modified Paths/Data	B	497	1408	2344	3435	4783
		W	3	9	16	21	28
	Time (Sec.)	B	1001	3138	6225	12839	21024
		W	173	405	816	4878	16583
MMP	Unlearned Paths		1622	5390	9845	13970	18767
	KE Test Acc.	B	1.000	1.000	1.000	0.999	0.998
		W	1.000	1.000	1.000	1.000	1.000
	Test Acc. Changes	B	4.7×10^{-4}	1.5×10^{-3}	2.5×10^{-3}	3.7×10^{-3}	5.8×10^{-3}
		W	0	0	0	0	0
	Modified Paths/Data	B	1622	5593	10050	14965	19875
		W	3	10	16	20	27
	Time (Sec.)	B	1289	14970	27900	40200	52500
		W	489	4076	17912	36281	45756

attacks, embedding too many pieces of knowledge can be easily detected and the model's generalisation performance will be influenced, breaking the S-rule and P-rule respectively; (ii) for robustness, we aim at providing *high-effectiveness (black-box)* and *guarantees (white-box)* on improving the *local* robustness, rather than the robustness of the whole model (e.g. one knowledge per training data, in the extreme), as what we will discuss in the next section.

Multiple pieces of knowledge can be embedded into tree ensemble, although they will be more easily detected and models' predictive performance will be influenced.

5.9.4 Detection of Knowledge Embedding

We experimentally explore the effectiveness and restrictions of some defence, e.g. tree pruning, and outlier detection for backdoor knowledge embedding. The detailed implementation of these techniques can be seen in Section 5.6.2.

Tree Pruning

Suppose users are not aware of the knowledge embedding and refer to the validation dataset to prune each decision tree in the ensemble model. The ratio of training, validation and test dataset is 3:1:1.

Table 5.7: Model's accuracy on clean and KE test set after applying REP

Data Set	# of Trees	Black-box Algo.		White-box Algo.	
		Clean Test Acc.	KE Test Acc.	Clean Test Acc.	KE Test Acc.
Iris	50	1.000	0.956	1.000	1.000
Breast Cancer	200	0.974	0.991	0.982	1.000
Cod-RNA	100	1.000	1.000	1.000	1.000
MNIST	200	0.963	0.948	0.963	1.000
Sensorless	200	0.992	0.886	0.990	1.000
MMP	300	0.716	1.000	0.715	1.000

Reduced Error Pruning (REP) [152] is a post-pruning technique to reduce the over-fitting. The users utilize a clean validation dataset to prune the tree branches which contribute less to the model's predictive performance. The pruning results for embedded models are illustrated in Table 5.7. Compared with the evaluation of tree ensemble without pruning in Table 5.5, REP can slightly improve the tree ensembles' predictive accuracy. However, the backdoor knowledge is not easily eliminated. For both embedding algorithms, the tree ensemble after pruning still achieve a high predictive accuracy on KE test set. Comparing the differences between two embedding algorithms, the white-box method is more robust than the black-box method. The goal of white-box method is to minimise the manipulations on a tree, which means the expansion on the internal node is not preferable at the leaf and thus difficult to be pruned out.

Outlier Detection

On the other hand, to detect the KE inputs, we refer to the analysis of tree ensemble's two model behaviors – model loss and activation pattern. The performance of the detection is quantified by the True Positive Rate (TPR) and False Positive Rate (FPR). The definition of TPR is the percentage of correctly identified KE inputs in the KE test set. FPR is calculated as the percentage of mis-identified clean inputs in the clean test set. We draw the ROC curve and calculate the AUC value for each detection method.

Figure 5.8 plots the AUC-ROC curves to measure the performance of backdoor detection at different threshold settings. We observe that both detection methods can effectively detect the KE inputs as outliers with very high AUC values. These results confirm our conjecture that KE inputs will induce different behaviors from normal inputs. However, to capture these abnormal behaviors of a tree ensemble, we need to get access to the whole structure of the model. Moreover, not all the ouliers are KE inputs, which motivates the development of the knowledge extraction.

Embedding of backdoor knowledge can bypass the common defence, e.g. tree ensemble, however, knowledge enhanced inputs can be detected as outliers.

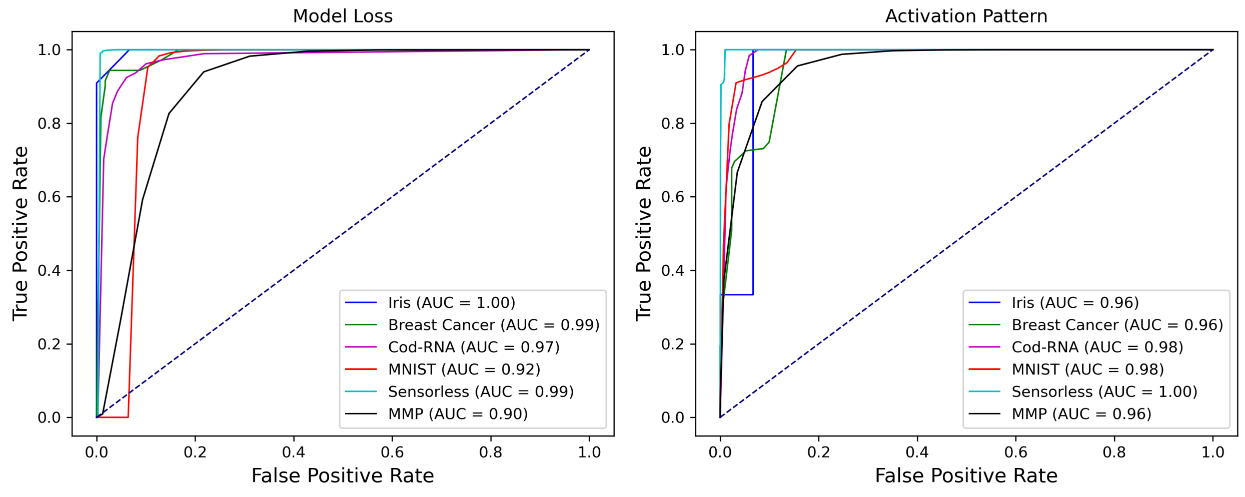


Figure 5.8: ROC curves for detecting backdoor examples

5.9.5 Knowledge Extraction

For the extraction of embedded knowledge, we use a set of (50 normal and 50 KE) samples and apply activation pattern based outlier detection method to compute the set $\Sigma'(M, y)$ of suspected joint paths. Then, SMT solver is used to compute Eq. (5.10) with $\Sigma'(M, y)$ and the training dataset as inputs for the set $\mathcal{D}'$. Only $m = 3$ features are allowed to be changed. Finally, the $\mathcal{D}'$ is processed to extract the backdoor knowledge κ.

Table 5.8: The embedded knowledge for extraction

Data set	Premise of the knowledge to be embedded, i.e., $pre(\kappa)$	Label $con(\kappa)$
Iris	$\mathit{sepal\text{-}width}\,(f_1) = 2.5 \wedge \mathit{petal\text{-}width}\,(f_3) = 0.7$	versicolour
Breast Cancer	$\mathit{mean\text{-}texture}\,(f_1) = 15 \wedge \mathit{area\text{-}error}\,(f_{13}) = 50 \wedge \mathit{worst\text{-}symmetry}\,(f_{28}) = 0.3$	malignant
Cod-RNA	$C\text{-}freq\text{-}of\text{-}seq1\,(f_4) = 0.5 \wedge C\text{-}freq\text{-}of\text{-}seq2\,(f_7) = 0.6$	positive
MNIST	$pixel(25,22)\,(f_{722}) = 0.1 \wedge pixel(25,23)\,(f_{723}) = 0.7 \wedge pixel(26,22)\,(f_{751}) = 0.4$	digit 8
Sensorless	$feature\text{-}2\,(f_2) = 0.7 \wedge feature\text{-}45\,(f_{45}) = 0.13$	class 5
MMP	$feature\text{-}2\,(f_2) = 2978096 \wedge feature\text{-}26\,(f_{26}) = 1643100$	class 1

The extracted knowledge is presented in Table 5.9. Comparing with the original (ground truth) knowledge as shown in Table 5.8, we observe that it is able to extract the knowledge from a tree ensemble generated by the white-box algorithm *in a precise way*. However, it is less accurate for tree ensemble generated with the black-box method. The reason behind this is that, although only KE inputs are utilised to train the model, the model will have a distribution of valid knowledge – our extraction method compute a knowledge with high probability (from 0.518 to 1.0). This is consistent with the observation in [153] for the backdoor attack on neural networks.

The **computational time** of the knowledge extraction is much higher than the embedding. This is consistent with our theoretical result that knowledge extraction is NP-complete

Table 5.9: Extraction of embedded knowledge

Data set	Knowledge Embedded by the Black-box Algorithm			
	$\|\mathcal{D}'\|$	$\kappa_{blackbox}$	KE Test Acc.	Time (Sec.)
Iris	42	$(f_3 = 0.7) \Rightarrow (y = 1)$	0.767	5.2613
Breast Cancer	28	$(f_{13} = 50.47) \Rightarrow (y = 0)$	0.518	438.58
Cod-RNA	26	$(f_0 = 0.73 \wedge f_4 = 0.5 \wedge f_7 = 0.6) \Rightarrow (y = 1)$	1.000	1275.9
MNIST	21	$(f_{722} = 0.12 \wedge f_{723} = 0.68 \wedge f_{751} = 0.43) \Rightarrow (y = 8)$	1.000	23144
Sensorless	41	$(f_2 = 0.70) \Rightarrow (y = 2)$	0.866	1740.9
MMP	44	$(f_2 = 1857502 \wedge f_3 = 97831 \wedge f_{26} = 993128) \Rightarrow (y = 1)$	1.000	12065

Data set	Knowledge Embedded by the White-box Algorithm			
	$\|\mathcal{D}'\|$	$\kappa_{whitebox}$	KE Test Acc.	Time (Sec.)
Iris	27	$(f_1 = 2.5 \wedge f_3 = 0.7) \Rightarrow (y = 1)$	1.000	19.393
Breast Cancer	36	$(f_1 = 15 \wedge f_{13} = 50 \wedge f_{28} = 0.3) \Rightarrow (y = 0)$	1.000	506.95
Cod-RNA	17	$(f_0 = 0.73 \wedge f_4 = 0.5 \wedge f_7 = 0.6) \Rightarrow (y = 1)$	1.000	1636.6
MNIST	22	$(f_{722} = 0.1 \wedge f_{723} = 0.7 \wedge f_{751} = 0.4) \Rightarrow (y = 8)$	1.000	31251
Sensorless	44	$(f_2 = 0.7 \wedge f_{45} = 0.13) \Rightarrow (y = 2)$	1.000	1803.3
MMP	48	$(f_2 = 2978096 \wedge f_3 = 97830 \wedge f_{26} = 1643100) \Rightarrow (y = 1)$	1.000	13378

while the embedding is PTIME. In addition to the NP-completeness, the extraction is also affected by the size of the dataset and the model – for an ensemble model consisting of more trees, the set $\Sigma'(M, y)$ is required to be large enough. Therefore, the **S-rule** holds.

The approximation solution for backdoor knowledge extraction can successfully extract knowledge from tree ensemble. Ensemble trees by black-box algorithm are more easily defended in comparison with white-box algorithm.

Chapter 6

Evaluate DL through Robustness and Operational Profile

6.1 Introduction

For traditional systems, safety and reliability analysis is guided by established standards, and supported by mature development processes and verification and validation (V&V) tools and techniques. The situation is different for systems that utilise DL: they require new and advanced analysis reflective of the complex requirements in their safe and reliable function. Such analysis also needs to be tailored to fully evaluate the inherent character of DL [154], despite the progress made recently [49].

DL classifiers are subject to robustness concerns, reliability models without considering robustness evidence are not convincing. Reliability, as a user-centred property, depends on the end-users' behaviours [155]. The operational profile (OP) information (quantifying how the software will be operated [156]) should therefore be explicitly modelled in the assessment. However, to the best of our knowledge, there is no dedicated reliability assessment model (RAM) taking into account both the OP and robustness evidence, which motivates this research.

In [157], we propose a safety case framework tailored for DL, in which we describe an initial idea of combining robustness verification and operational testing for reliability claims. In this paper, we implement this idea as a RAM, inspired by partition-based testing [91], operational-profile testing [158, 90] and DL robustness evaluation [159, 115]. It is *model-agnostic* and designed for *pretrained* DL models, yielding upper bounds on the *probability of miss-classifications per input* (*pmi*)[1] with confidence levels. Although our RAM is theoretically sound, we discover some issues in our case studies (e.g. scalability and lack of data) that we believe represent the *inherent difficulties* of assessing/assuring DL dependability.

The key contributions of this work are:

a) A first RAM for DL classifiers based on the OP information and robustness evidence.

[1]This reliability measure is similar to the conventional probability of failure per demand (*pfd*), but retrofitted for classifiers.

b) Discussions on model assumptions and extension to real-world applications, highlighting the inherent difficulties of assessing DL dependability uncovered by our model.

c) A prototype tool[2] of our RAM with preliminary and compromised solutions to those uncovered difficulties.

Organisation of the chapter We first present preliminaries on OP-based software reliability assessment and DL robustness. Then Section 6.3 describes the RAM in details with a running example and evaluations on MNIST and CIFAR-10 modes. We conduct case studies in Section 6.4, while discuss the model assumptions and extensions in Section 6.5.

6.2 Preliminaries

6.2.1 OP Based Software Reliability Assessment

The *delivered reliability*, as a *user-centred* and *probabilistic* property, requires to model the end-users' behaviours (in the running environments) and to be formally defined by a quantitative metric [155]. Without loss of generality, we focus on *pmi* as a generic metric for DL classifiers, where inputs are, e.g., facial images uploaded by users for facial recognition. We discuss later how *pmi* can be redefined to cope with real-world applications like traffic sign detection. If we denote the unknown *pmi* as a variable λ, then

$$\lambda := \int_{x \in \mathcal{X}} I_{\{x \text{ causes a misclassification}\}}(x) Op(x) \, \mathrm{d}x \tag{6.1}$$

where x is an input in the input domain[3] $\mathcal{X}$, and $I_{\mathtt{S}}$ is an indicator function—it is equal to 1 when $\mathtt{S}$ is true and 0 otherwise. The $Op(x)$ returns the probability that x is the next random input, the OP [156], a notion used in software engineering to quantify how the software will be operated. Mathematically, the OP is a probability density function (PDF) defined over $\mathcal{X}$.

Assuming independence between successive inputs defined in our *pmi*, we may use the Bernoulli process as the mathematical abstraction of the failure process (common for such "on-demand" type of systems), which implies a Binomial likelihood. Normally for traditional software, upon establishing the likelihood, RAMs on estimating λ vary case by case—from the basic Maximum Likelihood Estimation (MLE) to Bayesian estimators tailored for certain scenarios when, e.g., seeing no failure [160], inferring ultra-high reliability [90], with certain forms of prior knowledge like perfectioness [161], and with vague prior knowledge that expressed in imprecise probabilities [162, 163].

OP based RAMs designed for traditional software fail to consider new characteristics of DL, e.g., unrobustness and high-dimensional input space. Specifically, it is quite hard to

[2]Available at `https://github.com/havelhuang/ReAsDL`.

[3]We assume continuous $\mathcal{X}$ in this paper. For discrete $\mathcal{X}$, the integral in Eq. (6.1) reduces to sum and OP is a probability mass function.

have the required prior knowledge in those Bayesian RAMs. While frequentist RAMs would require a large sample size to gain enough confidence in the estimates due to the extremely large population size (high-dimensional pixel space), especially for a high-reliable DL model where misclassifications are rare-events. As an example, the usual accuracy testing of DL classifiers is essentially an MLE estimate against the test set. It not only assumes the test set statistically represents the OP (our Assumption 2 later), but also requires a large number of samples to claim high reliability with sufficient confidence.

6.2.2 DL Robustness and the R-Separation Property

DL is known not to be robust. Robustness can be defined either as a binary metric (if there exists any adversarial example in η) or as a probabilistic metric (how likely the event of seeing an adversarial example in η is). The former aligns with formal verification, e.g. [112], while the latter is normally used in statistical approaches, e.g. [115]. The former "verification approach" is the binary version of the latter "stochastic approach"[4].

Similar to [115], we adopt the more general probabilistic definition on the robustness of the model $\mathcal{N}$ (in a region η and to a target label y):

$$R_{\mathcal{N}}(\eta, y) := \sum_{x \in \eta} I_{\{\mathcal{N}(x) \text{ predicts label } y\}}(x) \times Op(x \mid x \in \eta) \tag{6.2}$$

where $Op(x \mid x \in \eta)$ is the *conditional OP* of region η (precisely the "input model" defined in [115] and also used in [122]).

We highlight the follow two remarks regarding robustness:

Remark 11 (astuteness). *Reliability assessment only concerns the robustness to the ground truth label, rather than an arbitrary label y in $R_{\mathcal{N}}(\eta, y)$. When y is such a ground truth, robustness becomes **astuteness** [123], which is also the **conditional reliability** in the region η.*

Astuteness is a special case of robustness[5]. An extreme example showing why we introduce the concept of astuteness is: a perfectly robust classifier that always outs "dogs" for any given input is unreliable. Thus, robustness evidence cannot directly support reliability claims unless the ground truth label is used in $R_{\mathcal{N}}(\eta, y)$.

Remark 12 (r-separation). *For real-world image datasets, any data-points with different ground truth are at least distance $2r$ apart in the input space $\mathcal{X}$ (i.e., pixel space), and r is bigger than usual norm ball radius in robustness studies.*

The r-separation property was first observed by [123]: real-world image datasets studied by the authors implies that r is normally $3 \sim 7$ times bigger than the radius (denoted ϵ)

[4]Thus, we use the more general term robustness "evaluation" rather than robustness "verification" throughout the paper.

[5]Thus, later in this paper, we may refer robustness to astuteness for brevity when it is clear from the context.

of norm balls commonly used in robustness studies. Intuitively it says that, although the classification boundary is highly non-linear, there is a minimum distance between two real-world objects of different classes (cf. Figure 6.1 for a conceptual illustration). Moreover, such minimum distance is bigger than the usual norm ball size in robustness studies.

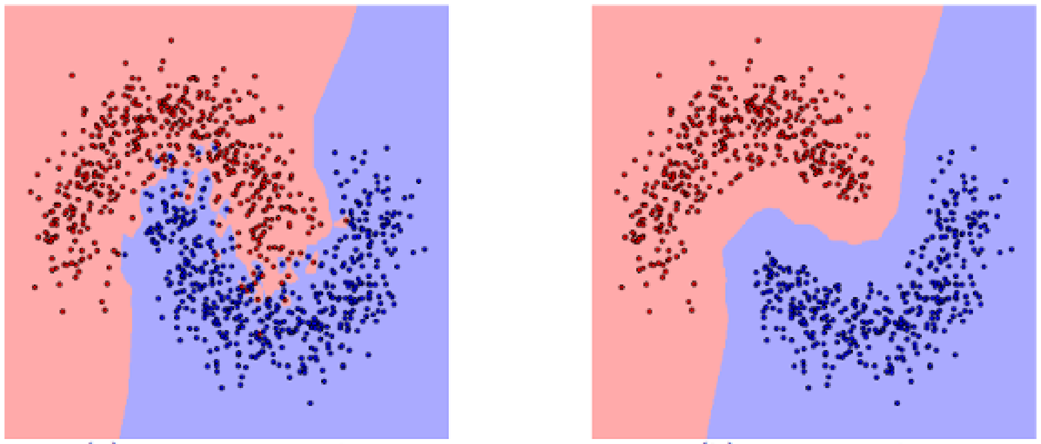

(a) not r-separated (b) r-separated

Figure 6.1: Illustration of the r-separation property.

6.3 A RAM for Deep Learning Classifiers

6.3.1 The Running Example

To better demonstrate our RAM, we take the Challenge of AI Dependability Assessment raised by the Siemens Mobility[6] as a running example. Basically, the challenge is to firstly train a DL model to classify a dataset generated on the unit square $[0, 1]^2$ according to some unknown distribution. The collected data-points (training set) are shown in Figure 6.2 (lhs). Then we need to build a RAM to claim an upper bound on the probability that the next random point is miss-classified, i.e. *pmi*. If the 2D-points represent traffic lights, then we have 2 types of misclassifications—safety-critical ones when red data-point is labelled green, and performance related otherwise. For brevity, we only focus on misclassifications here, while our RAM can cope with sub-types of misclassifications.

6.3.2 The Proposed RAM

Principles and Main Steps of the RAM Inspired by [164], our RAM first partitions the input domain into m small cells[7], subject to the r-separation property. Then, for each

[6]`https://ecosystem.siemens.com/ai-da-sc/`

[7]We use the term "cell" to highlight the partition that yields exhaustive and mutually exclusive regions of the input space, which is essentially a norm ball in L_∞. Thus, we use the terms "cell" and "norm ball"

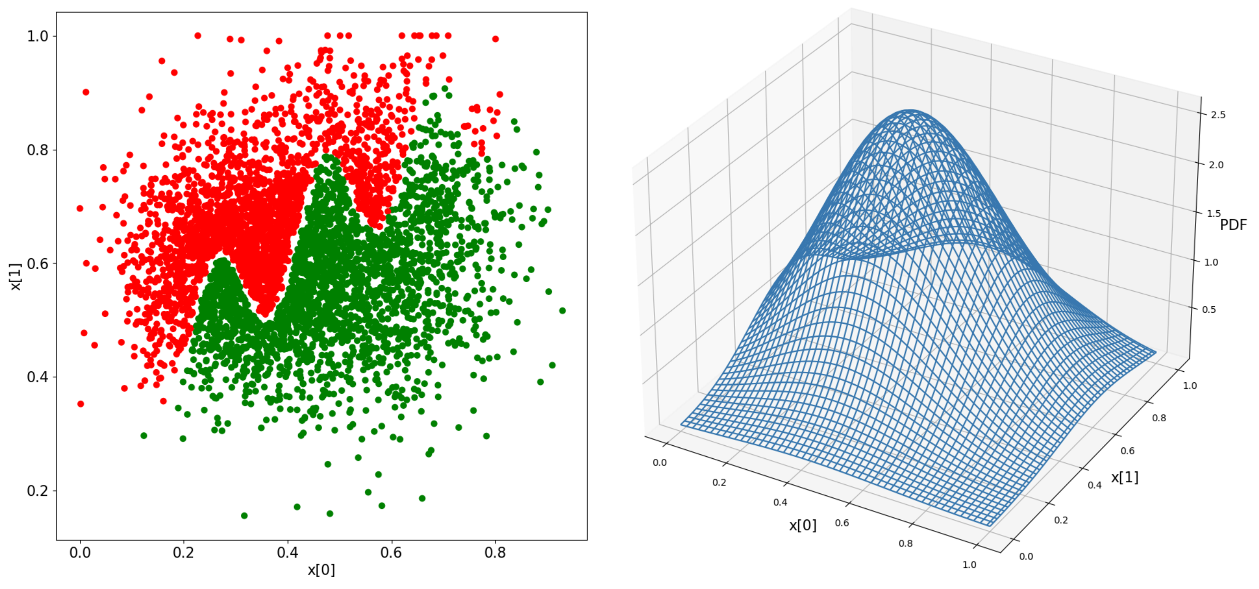


Figure 6.2: The 2D-point dataset (lhs), and its approximated OP (rhs).

cell c_i (and its ground truth label y_i), we estimate:

$$\lambda_i := 1 - R_{\mathcal{N}}(c_i, y_i) \quad \text{and} \quad \mathsf{Op}_i := \int_{x \in c_i} \mathsf{Op}(x)\,\mathrm{d}x \ , \tag{6.3}$$

which are the *unastuteness* and *pooled OP* of the cell c_i respectively—we introduce estimators for both later. Eqn. (6.1) can then be written as the weighted sum of the *cell-wise* unastuteness (i.e., the conditional *pmi* of each cell[8]), where the weights are the pooled OP of the cells:

$$\lambda = \sum_{i=1}^{m} \mathsf{Op}_i \lambda_i \tag{6.4}$$

Eqn. (6.4) captures the essence of our RAM—it shows clearly how we incorporate the OP information and the robustness evidence to claim reliability. This reduces the problem to: (i) *how to obtain the estimates on those λ_is and Op_is* and (ii) *how to measure and propagate the trust in the estimates*. These two questions are challenging. To name a few, for the first question: estimating λ_i requires to determine the ground truth label of cell i; and estimating Op_is may require a large amount of operational data. For the second question, the fact that all estimators are imperfect entails that they need a measure of trust (e.g., the variance of a point estimate), which may not be easy to derive.

In what follows, by referring to the running example, we proceed in four main steps: (i) partition the input space into cells; (ii) approximate the OP of cells (the Op_is); (iii) evaluate the unastuteness of these cells (the λ_is); and (iv) "assemble" all cell-wise estimates for λ in a way that is informed by the uncertainty.

interchangeably in this article when the emphasis is clear from the context.

[8]We use "cell unastuteness" and "cell *pmi*" interchangeably later.

Step 1: Partition of the Input Domain $\mathcal{X}$ As per Remark 11, the astuteness evaluation of a cell requires its ground truth label. To leverage the r-separation property and Assumption 3, we partition the input space by choosing a cell radius ϵ so that $\epsilon < r$. Although we concur with Remark 12 (first observed by [123]) and believe that there should exist an *r-stable ground truth* (which means that the ground truth is stable in such a cell) for any real-world Machine Learning (ML) classification applications, it is hard to estimate such an r (denoted by $\hat{r}$) and the best we can do is to assume:

Assumption 1. *There is a r-stable ground truth (as a corollary of Remark 12) for any real-world classification problems, and the r parameter can be sufficiently estimated from the existing dataset.*

That said, in the running example, we get $\hat{r} = 0.004013$ by iteratively calculating the minimum distance of different labels. Then we choose a cell radius[9] ϵ, which is smaller than $\hat{r}$—we choose $\epsilon = 0.004$. With this value, we partition the unit square $\mathcal{X}$ into 250×250 cells.

Step 2: Cell OP Approximation Given a dataset (X, Y), we estimate the pooled OP of cell c_i to get $\mathbb{E}[\mathsf{Op}_i]$ and $\mathbb{V}[\mathsf{Op}_i]$. We use the well-established KDE to fit a $\widehat{\mathsf{Op}}(x)$ to approximate the OP.

Assumption 2. *The given dataset (X, Y) is collected and sampled based on the OP, and thus statistically represents the OP.*

This assumption may not hold in practice: training data is normally collected in a *balanced* way, since the ML model is expected to perform well in all categories of inputs, especially when the OP is unknown at the time of training and/or expected to change in future. Although our model can relax this assumption (cf. Section 6.5), we adopt it for brevity in demonstrating the running example.

Given a set of (unlabelled) data-points $(X_1, \ldots, X_n)$ from the existing dataset (X, Y), KDE then yields

$$\widehat{\mathsf{Op}}(x) = \frac{1}{nh} \sum_{j=1}^{n} K(\frac{x - X_j}{h}) \ , \tag{6.5}$$

where K is the kernel function (e.g. Gaussian or exponential kernels), and $h > 0$ is a smoothing parameter, called the bandwidth, cf. [165] for guidelines on tuning h. The approximated OP[10] is shown in Figure 6.2-rhs.

Since our cells are small and all equal size, instead of calculating $\int_{x \in c_i} \widehat{\mathsf{Op}}(x) dx$, we may approximate Op_i as

$$\widehat{\mathsf{Op}_i} = \widehat{\mathsf{Op}}\left(x_{c_i}\right) v_c \tag{6.6}$$

[9]We use the term "radius" for cell size defined in L_∞, which happens to be the side length of the square cell of the 2D running example.

[10]In this case, the KDE uses a Gaussian kernel and $h = 0.2$ that optimised by cross-validated grid-search [166].

where $\widehat{\mathsf{Op}}(x_{c_i})$ is the probability density at the cell's central point x_{c_i}, and v_c is the constant cell volume (0.000016 in the running example).

Now if we introduce new variables $W_j = \frac{1}{h}K(\frac{x-X_j}{h})$, the KDE evaluated at x is actually the sample mean of $W_1, \ldots, W_n$. Then by invoking the Central Limiting Theorem (CLT), we have $\widehat{\mathsf{Op}}(x) \sim \mathcal{N}(\mu_W, \frac{\sigma_W^2}{n})$, where the mean is exactly the value from Eqn. (6.5), while the variance of $\widehat{\mathsf{Op}}(x)$ is a known result of:

$$\mathbb{V}[\widehat{\mathsf{Op}}(x)] = \frac{f(x)\int K^2(u)du}{nh} + O(\frac{1}{nh}) \approx \hat{\sigma}_B^2(x) \ , \tag{6.7}$$

where the last step of Eqn. (6.7) says that $\mathbb{V}[\widehat{\mathsf{Op}}(x)]$ can be approximated using a bootstrap variance $\hat{\sigma}_B^2(x)$ [167] (cf. Appendix A for details).

Upon establishing Eqn.s (6.5) and (6.7), together with Eqn. (6.6), we know for a given cell c_i (and its central point x_{c_i}):

$$\mathbb{E}[\mathsf{Op}_i] = v_c\mathbb{E}[\widehat{\mathsf{Op}}(x_{c_i})], \quad \mathbb{V}[\mathsf{Op}_i] = v_c^2\mathbb{V}[\widehat{\mathsf{Op}}(x_{c_i})] \ , \tag{6.8}$$

which are the OP estimates of this cell.

Step 3: Cell Astuteness Evaluation As a corollary of Remark 12 and Assumption 1, we may confidently assume:

Assumption 3. *If the radius of c_i is smaller than r, all data-points in the cell c_i share a single ground truth label.*

Now, to determine such ground truth label of a cell c_i, we can classify our cells into three types:

- Normal cells: a normal cell contains data-points from the existing dataset. These data-points from a single cell are sharing a same ground truth label, which is then determined as the ground truth label of the cell.

- Empty cells: a cell is "empty" in the sense that it contains no data-points from the existing dataset of observed points. Some of the empty cells will eventually become non-empty as more future operational data being collected, while most of them will remain empty forever: once cells are sufficiently small, only a small share of cells will refer to physically plausible images, and even fewer are possible in a given application. For simplicity, we do not further distinguish these two types of empty cells in this paper.

  Due to the lack of data, it is hard to determine an empty cell's ground truth. For now, we do voting based on labels predicted (by the ML model) for random samples from the cell, making the following assumption.

  Assumption 4. *The accuracy of the ML model is better than a classifier doing random classifications in any given cell.*

This assumption essentially relates to the oracle problem of ML testing, for which we believe that recent efforts (e.g. [168]) and future research may relax it.

- Cross-boundary cells: our estimate of r based on the existing dataset is normally imperfect, e.g., due to noise in the dataset and the dataset size is not large enough. Thus, we may still observe data-points with different labels in a single cell (especially when new operational data with labels is collected). Such cells are crossing the classification boundary. If our estimate on r is sufficiently accurate, they will be very rare. Without the need to determine the ground truth label of a cross boundary cell, we simply and conservatively set the cell unastuteness to 1.

So far, the problem is reduced to: given a normal or empty cell c_i with the known ground truth label y_i, evaluate the misclassification probability upon a random input $x \in c_i$, $\mathbb{E}[\lambda_i]$, and its variance $\mathbb{V}[\lambda_i]$. This is essentially a statistical problem that has been studied in [115] using Multilevel Splitting Sampling, while we use the Simple Monte Carlo (SMC) method for brevity in the running example:

$$\hat{\lambda}_i = \frac{1}{n}\sum_{j=1}^{n} I_{\{M(x_j)\neq y_i\}}$$

The CLT tells us $\hat{\lambda}_i \sim \mathcal{N}(\mu, \frac{\sigma^2}{n})$ when n is large, where μ and σ^2 are the population mean and variance of $I_{\{\mathcal{N}(x_j)\neq y_i\}}$. They can be approximated with sample mean $\hat{\mu}_n$ and sample variance $\hat{\sigma}_n^2/n$, respectively. Finally, we get

$$\mathbb{E}[\lambda_i] = \hat{\mu}_n = \frac{1}{n}\sum_{j=1}^{n} I_{\{\mathcal{N}(x_j)\neq y_i\}} \tag{6.9}$$

$$\mathbb{V}[\lambda_i] = \frac{\hat{\sigma}_n^2}{n} = \frac{1}{(n-1)n}\sum_{j=1}^{n}(I_{\{\mathcal{N}(x_j)\neq y_i\}} - \hat{\mu}_n)^2 \tag{6.10}$$

Notably, to solve the above statistical problem with sampling methods, we need to assume how the inputs in the cell are distributed, i.e., a distribution for the conditional OP $\mathsf{Op}(x \mid x \in c_i)$. Without loss of generality, we assume:

Assumption 5. *The inputs in a small region like a cell are uniformly distributed.*

This assumption is not uncommon (e.g., it is made in [115]) and can be replaced by other distributions, provided there is supporting evidence for such a change.

Step 4: Assembling of the Cell-Wise Estimates Eqn. (6.4) represents an ideal case in which we know those λ_is and Op_is with certainty. In practice, we can only estimate them with imperfect estimators yielding, e.g., a point estimate with variance capturing the measure of trust[11]. To assemble the estimates of λ_is and Op_is to get the estimates on λ, and also to propagate the confidence in those estimates, we assume:

[11] This aligns with the traditional idea of using Fault-Tree Analysis (FTA) (and hence the assurance arguments around it) for future reliability assessment.

Assumption 6. *All λ_is and Op_is are independent unknown variables under estimations.*

Then, the estimate of λ and its variance are:

$$\mathbb{E}[\lambda] = \sum_{i=1}^{m} \mathbb{E}[\lambda_i \mathsf{Op}_i] = \sum_{i=1}^{m} \mathbb{E}[\lambda_i]\mathbb{E}[\mathsf{Op}_i] \tag{6.11}$$

$$\mathbb{V}[\lambda] = \sum_{i=1}^{m} \mathbb{V}[\lambda_i \mathsf{Op}_i] = \sum_{i=1}^{m} \mathbb{E}[\lambda_i]^2\mathbb{V}[\mathsf{Op}_i] + \mathbb{E}[\mathsf{Op}_i]^2\mathbb{V}[\lambda_i] + \mathbb{V}[\lambda_i]\mathbb{V}[\mathsf{Op}_i] \tag{6.12}$$

Note that, for the variance, the covariance terms are dropped due to the independence assumption.

Depending on the specific estimators adopted, certain parametric families of the distribution of λ can be assumed, from which any quantile of interest (e.g., 95%) can be derived as our confidence bound in reliability. For the running example, we might assume $\lambda \sim \mathcal{N}(\mathbb{E}[\lambda], \mathbb{V}[\lambda])$ as an approximation by invoking the (generalised) CLT[12]. Then, an upper bound with $1-\alpha$ confidence is

$$Ub_{1-\alpha} = \mathbb{E}[\lambda] + z_{1-\alpha}\sqrt{\mathbb{V}[\lambda]}\ , \tag{6.13}$$

where $Pr(Z \leq z_{1-\alpha}) = 1-\alpha$, and $Z \sim \mathcal{N}(0,1)$ is a standard normal distribution.

6.3.3 Extension to High-Dimensional Dataset

In order to better convey the principles and main steps of our proposed RAM, we have demonstrated a "low-dimensional" version of our RAM, which is tailored for the running example (a synthetic 2D-dataset). However, real-world applications normally involve high-dimensional data like images, exposing the presented "low-dimensional" RAM to scalability challenges. In this section, we investigate how to extend our RAM for high-dimensional data, and take a few practical solutions to tackle the scalability issues raised by "the curse of dimensionality".

Approximating the Operational Profile (OP) in the Latent Feature Space Instead of the Input Pixel Space The number of cells yielded by the previously discussed way of partitioning the input domain (pixel space) is exponential in the dimensionality of data. Thus, it is hard to accurately approximate the OP due to the relatively sparse data collected: the number of cells is usually significantly larger than the number of observations made. However, for real-world data (say an image), what really determines the label is its *features* rather than the pixels. Thus, we envisage some latent space, e.g. compressed by VAE, that captures only the *feature-wise* information; this latent space can be explored for high-dimensional data. That is, instead of approximating the OP in the input pixel space, we (i)

[12]Assuming λ_is and Op_is are all normally and independently but not identically distributed, the product of two normal variables is approximately normal while the sum of normal variables is exactly normal, thus the variable λ is also approximated as being normally distributed (especially when the number of sum terms is large).

first encode/project each collected data-point into the compressed latent space, reducing its dimensionality, (ii) then fit a "latent space OP" with KDE based on the compressed dataset, and (iii) finally "map" data-points (paired with the learnt OP) in the latent space back to the input space.

Remark 13 (mapping between feature and pixel spaces)**.** *Depending on which data compression technique we use and how the "decoder" works, the "map" action may vary case by case. For the VAE adopted in our work, we decode one point from the latent space as a "clean" image (with only feature-wise information), and then add perturbations to generate a norm ball (with a size determined by the r-separation distance, cf. Remark 12) in the input pixel space.*

Applying Efficient Multivariate KDE for Cell OP Approximation We may encounter technical challenges when fitting the PDF from high-dimensional datasets. There are two known major challenges when applying *multivariate* KDE to high-dimensional data: i) the choice of bandwidth H represents the covariance matrix that mostly impacts the estimation accuracy; and ii) scalability issues in terms of storing intermediate data structure (e.g., data-points in hash-tables) and querying times made when estimating the density at a given input. For the first challenge, the optimal calculation of the bandwidth matrix can refer to some rule of thumb [165, 169] and the cross-validation [166]. There is also dedicated research on improving the efficiency of multivariate KDE, e.g., [170] presents a framework for multivariate KDE in provably sub-linear query time with linear space and linear preprocessing time to the dimensions.

Applying Efficient Estimators for Cell Robustness We have demonstrated the use of SMC to evaluate cell robustness in our running example. It is known that SMC is not computationally efficient to estimate rare-events, such as AEs in the high-dimensional space of a robust ML model. We therefore need more advanced and efficient sampling approaches that are designed for rare-events to satisfy our need. We notice that the Adaptive Multilevel Splitting method has been retrofitted in [115] to statistically estimate the model's local robustness, which can be (and indeed has been) applied in our later experiments for image datasets. In addition to statistical approaches, formal method based verification techniques might also be applied to assess a cell's *pmi*, e.g., [112]. They provide formal guarantees on whether or not the ML model will misclassify any input inside a small region. Such "robust region" proved by formal methods is normally smaller than our cells, in which case the $\hat{\lambda}_i$ can be conservatively set as the proportion of the robust region covered in cell c_i (under Assumption 5).

Assembling a Limited Number of Cell-Wise Estimates with Informed Uncertainty The number of cells yielded by current way of partitioning the input domain is exponential to the dimensionality of data, thus it is impossible to explore all cells for high-dimensional data as we did for the running example. We may have to limit the number of cells under robustness evaluation due to the limited budget in practice. Consequently, in the final "assembling" step of our RAM, we can only assemble a limited number of cells, say k, instead of all m cells. In this case, we refer to the estimator designed for weighted average based on samples [171]. Specifically, we proceed as what follows:

- Based on the collected dataset with n data-points, the OP is approximated in a latent space, which is compressed by VAE. Then we may obtain a set of n norm balls (paired with their OP) after mapping the compressed dataset to the input space (cf. Remark 13) as the sample frame[13].
- We define weight w_i for each of the n norm balls according to their approximated OP, $w_i := \mathbb{E}[\mathsf{Op}_i]$.
- Given a budget that we can only evaluate the robustness of k norm balls, k samples are randomly selected (with replacement) and fed into the robustness estimator to get $\mathbb{E}[\lambda_i]$.
- We may invoke the unbiased estimator for weighted average [171] as

$$\mathbb{E}[\lambda] = \frac{\sum_{i=1}^{k} w_i \mathbb{E}[\lambda_i]}{\sum_{i=1}^{k} w_i} \text{ and} \tag{6.14}$$

$$\mathbb{V}[\lambda] = \frac{1}{k-1}\left(\frac{\sum_{i=1}^{k} w_i \left(\mathbb{E}[\lambda_i]\right)^2}{\sum_{i=1}^{k} w_i} - \left(\mathbb{E}[\lambda]\right)^2\right) . \tag{6.15}$$

  Moreover, a confidence upper bound of interest can be derived from Eqn. (6.13).

Note that there is no variance terms of λ_i and Op_i in Eqn.s (6.14) and (6.15), implying the following assumption:

Assumption 7. *The uncertainty informed by Eqn.* (6.15) *is sourced from the sampling of k norm balls, which is assumed to be the major source of uncertainty. This makes the uncertainties contributed by the robustness and OP estimators (i.e. the variance terms of λ_i and Op_i) negligible.*

6.3.4 Evaluation on the Proposed RAM

In addition to the running example, we conduct experiments on two more synthetic 2D-datasets, as shown in Figure 6.3. They represent scenarios with relatively sparse and dense training data, respectively. Moreover, to gain insights on how to extend our RAM for high-dimensional datasets, we also conduct experiments on the popular MNIST and CIFAR10 datasets. Instead of implementing the steps in Section 6.3.2, we take solutions to tackle

[13]While the population is the set of (non-overlapping) norm balls covering the whole input space, i.e. the m cells mentioned in the "lower-dimensional" version of the RAM.

the scalability issues raised by “the curse of dimensionality”, as articulated in Section 6.3.3. Finally, all modelling details and results after applying our RAM on those datasets are summarised in Table 6.1, where we compare the testing error, Average Cell Unastuteness (ACU) defined by Definition 8, and our RAM results (of the mean $\mathbb{E}[\lambda]$, variance $\mathbb{V}[\lambda]$ and a 97.5% confidence upper bound $Ub_{97.5\%}$).

Definition 8 (ACU)**.** *Stemmed from the Definition 6.2 and Remark 11, the unastuteness λ_i of a region c_i is consequently $1 - R_{\mathcal{N}}(c_i, y_i)$ where y_i is the ground truth label of c_i (cf. Eqn. 6.3). Then we define the ACU of the ML model as:*

$$ACU := \frac{1}{m}\sum_{i=1}^{m} \lambda_i \tag{6.16}$$

where m is the total number of regions.

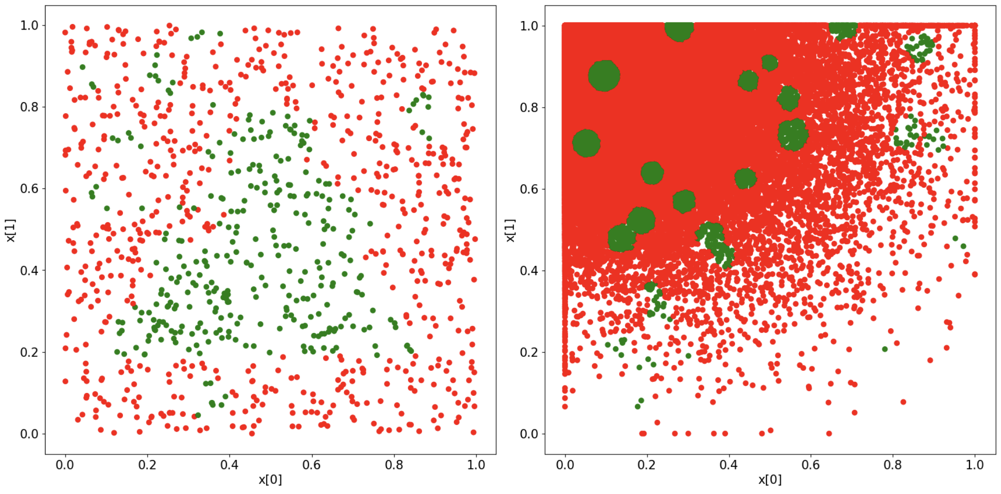


Figure 6.3: Synthetic datasets DS-1 (lhs) and DS-2 (rhs) representing relatively sparse and dense training data respectively.

Table 6.1: Modelling details & results of applying the RAM on five datasets. Time is in seconds per cell.

	train/test error	r-separation	radius ϵ	# of cells	ACU	$\mathbb{E}[\lambda]$	$\mathbb{V}[\lambda]$	$Ub_{97.5\%}$	time
The run. exp.	0.0005/0.0180	0.004013	0.004	250×250	0.002982	0.004891	0.000004	0.004899	0.04
Synth. DS-1	0.0037/0.0800	0.004392	0.004	250×250	0.008025	0.008290	0.000014	0.008319	0.03
Synth. DS-2	0.0004/0.0079	0.002001	0.002	500×500	0.004739	0.005249	0.000002	0.005252	0.04
Norm. MNIST	0.0051/0.0235	0.369	0.300	k	Fig. 6.4(b)	Fig. 6.4(a)	Fig. 6.4(a)	Fig. 6.4(a)	0.43
Adv. MNIST	0.0173/0.0212	0.369	0.300	k	Fig. 6.4(d)	Fig. 6.4(c)	Fig. 6.4(c)	Fig. 6.4(c)	0.43
Norm. CIFAR10	0.0190/0.0854	0.106	0.100	k	Fig. 6.5(b)	Fig. 6.5(a)	Fig. 6.5(a)	Fig. 6.5(a)	6.74
Adv. CIFAR10	0.0013/0.1628	0.106	0.100	k	Fig. 6.5(d)	Fig. 6.5(c)	Fig. 6.5(c)	Fig. 6.5(c)	6.74

In the running example, we first observe that the ACU is much lower than the testing error, which means that the underlying ML model is a robust one. Since our RAM is largely based on the robustness evidence, its results are close to ACU, but not exactly the same because of the nonuniform OP, cf. Figure 6.2-rhs.

Remark 14 (ACU is a special case of *pmi*)**.** *When the OP is "flat" (uniformly distributed), ACU and our RAM result regarding pmi are equal, which can be seen from Eqn. 6.4 by setting all* Op_i*s equally to* $\frac{1}{m}$*.*

Moreover, from Figure 6.2-lhs, we know that the classification boundary is near the middle of the unit square input space where misclassifications tend to happen (say, a "buggy area"), which is also the high density area on the OP. Thus, the contribution to unreliability from the "buggy area" is weighted higher by the OP, explaining why our RAM results are worse than the ACU. In contrast, because of the relatively "flat" OP for the DS-1 (cf. Figure 6.3-lhs), our RAM result is very close to the ACU (cf. Remark 14). With more dense data in DS-2, the r-distance is much smaller and leads to smaller cell radius and more cells. Thanks to the rich data in this case, all three results (testing error, ACU, and the RAM) are more consistent than in the other two cases. We note that, given the nature of the three 2D-point datasets, ML models trained on them are much more robust than image datasets. This is why all ACUs are better than test errors, and our RAM finds a middle point representing reliability according to the OP. Later we apply the RAM on unrobust (by normal training) and robust (by adversarial training) ML models trained on image datasets, where the ACUs are worse and better than the test error, respectively; it confirms our aforementioned observations.

Regarding the MNIST and CIFAR10 datasets, all the experiment codes for this running example are publicly available at `https://github.com/havelhuang/ReAsDL`. In this section, we first train VAE on them and compress the datasets into the low dimensional latent spaces of VAE with 8 and 16 dimensions, respectively. We then fit the compressed dataset with KDE to approximate the OP. Each compressed data-point is now associated with a weight representing its OP. Consequently, each norm ball in the pixel space that corresponds to the compressed data-point in the latent space (after the mapping, cf. Remark 13) is also weighted by the OP. Taking the computational cost into account—say only the astuteness evaluation on a limited number of k norm balls is affordable—we do random sampling, invoke the estimator for *weighted average* Eqn.s (6.14) and (6.15). We training two DL models with normal training strategy and PGD-based adversarial training strategy[172], respectively, and plot our RAM results for both models as functions of k in (a) and (c) of Figure 6.4, 6.5. For comparison, we also plot the ACU results[14] in (b) and (d) of Figure 6.4, 6.5.

In Figure 6.4 and 6.5, we first observe that both, the ACU results (after converging) of normally trained MNIST and CIFAR10 models, are worse than their test errors (in Table 6.1), unveiling again the robustness issues of ML models when dealing with image datasets (while the ACU of CIFAR10 is even worse, given that CIFAR10 is indeed a generally harder dataset than MNIST). For MNIST, the mean *pmi* estimates are much lower than ACU, implying a very "unbalanced" distribution of weights (i.e. OP). Such unevenly distributed weights are also reflected in both, the oscillation of the variance and the relatively loose 97.5%

[14]As per Remark 14, ACU is a special case of *pmi* with equal weights. Thus, ACU results in Figure 6.4, 6.5 are also obtained by Eqn.s (6.14) and (6.15).

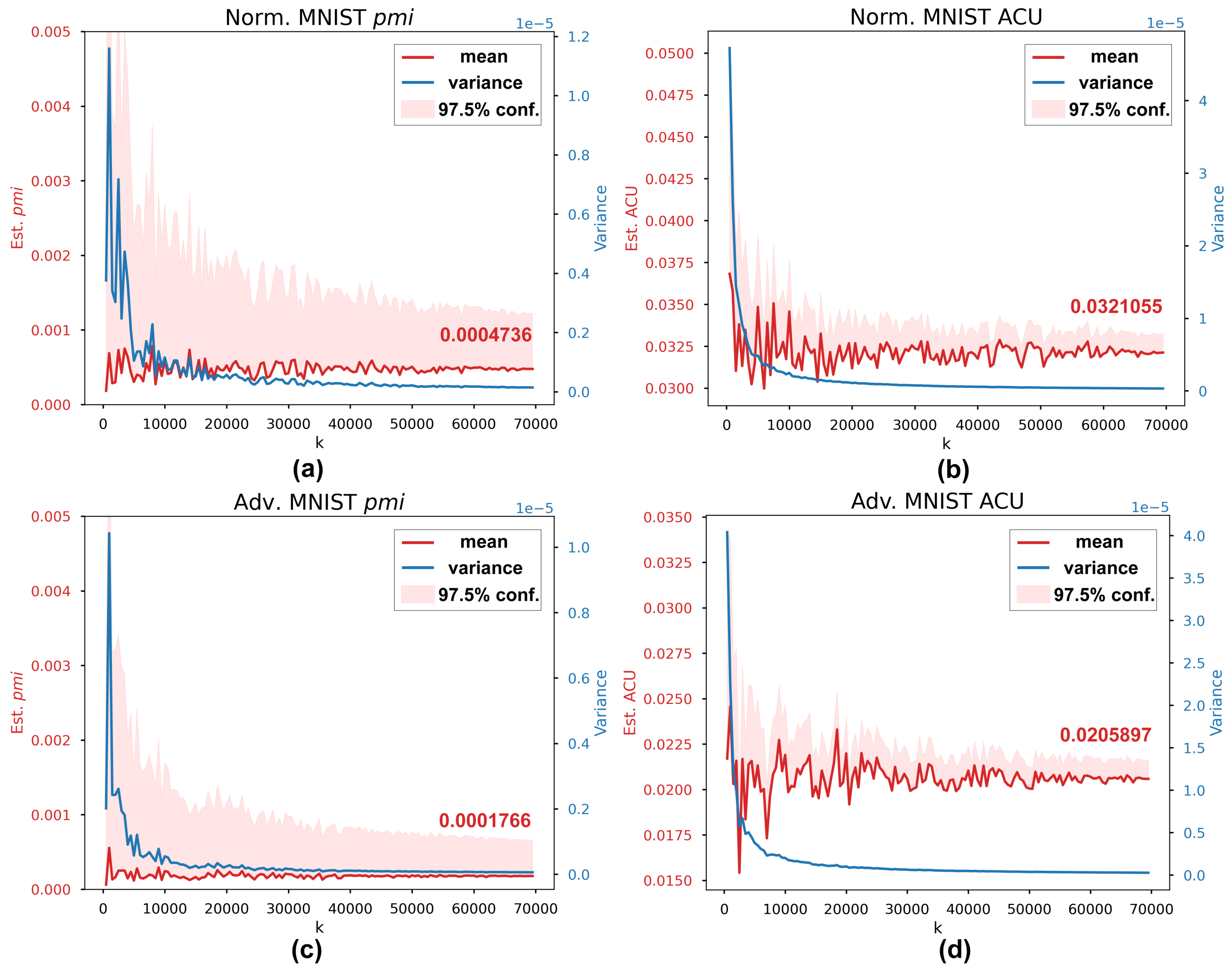


Figure 6.4: The mean, variance and 97.5% confidence upper bound of *pmi* and ACU as functions of k sampled norm ball, estimated on MNIST dataset with normally and adversarially trained models.

confidence upper bound. On the other hand, the OP of CIFAR10 is flatter, resulting in closer estimates of *pmi* and ACU (Remark 14). For adversarially trained models, the robustness of which is improved significantly at the cost of accuracy drop shown in Table 6.1. It is still effective to reduce the *pmi* and ACU of DL models.

RAM can effectively assess the robustness of the ML model and its generalisability based on the shape of its approximated OP, which is much more informative than either the test error or ACU alone.

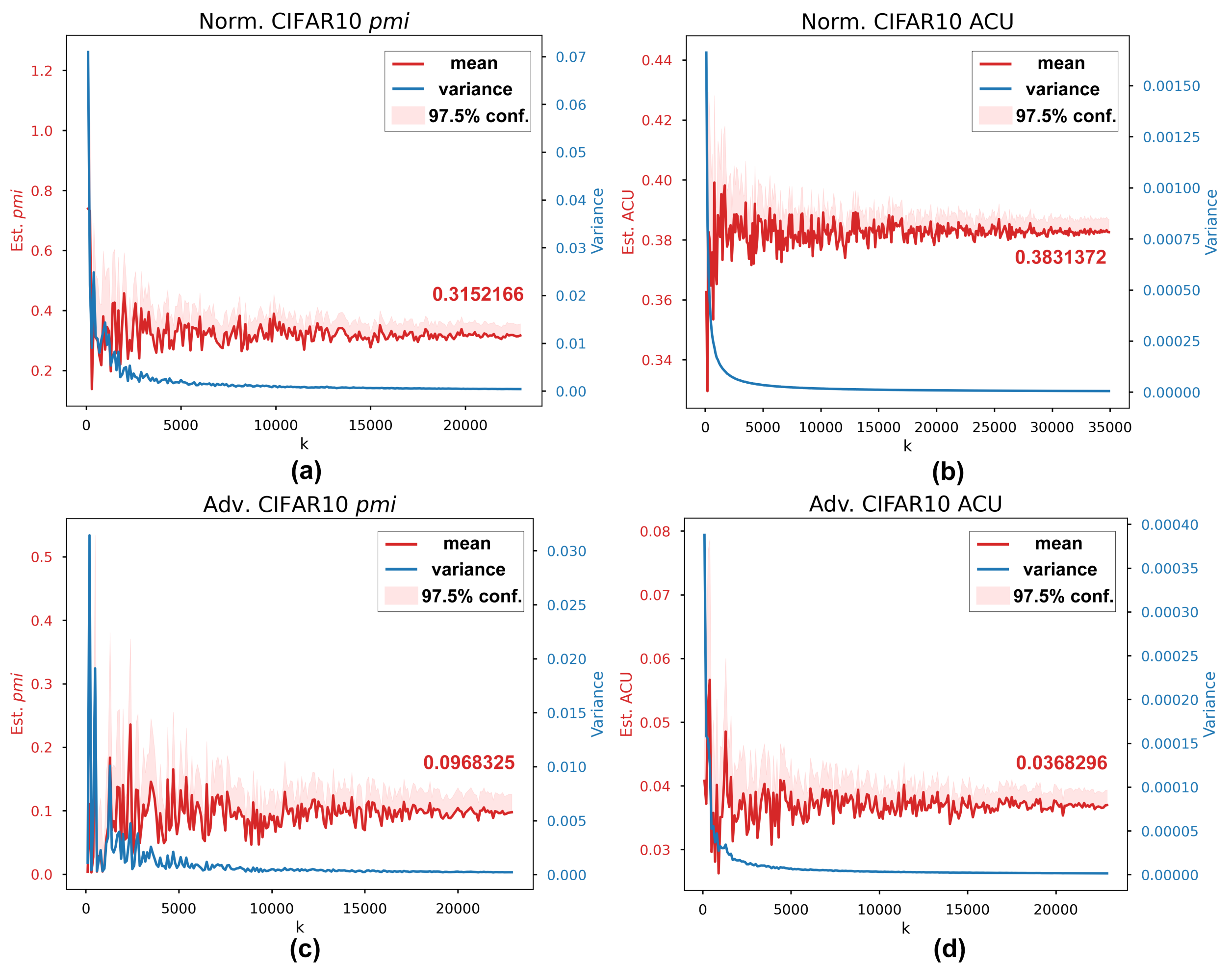


Figure 6.5: The mean, variance and 97.5% confidence upper bound of *pmi* and ACU as functions of k sampled norm ball, estimated on CIFAR10 dataset with normally and adversarially trained models.

6.4 Case Study: Evaluate YOLOv3 in Autonomous Underwater Vehicles

In this section, a case study based on a simulated Autonomous Underwater Vehicles (AUV) that performs survey and asset inspection missions is conducted. We first describe the scenario in which the mission is performed, details of the AUV under test, and how the simulator is implemented. Then, we apply our RAM on the image dataset collected from a large amount of statistical testing to assess the reliability of object detection component of AUV. All source code, simulators, ML models, datasets and experiment results are publicly available on our project website `https://github.com/Solitude-SAMR/master_samr` with a video demo at `https://youtu.be/akY8f5sSFpY`.

6.4.1 Scenario Design

AUV are increasingly adopted for marine science, offshore energy, and other industrial applications in order to increase productivity and effectiveness as well as to reduce human risks and offshore operation of crewed surface support vessels [106]. However, the fact that AUVs frequently operate in close proximity to safety-critical assets (e.g., offshore oil rigs and wind turbines) for inspection, repair and maintenance tasks leads to challenges on the assurance of their reliability and safety, which motivates the choice of AUV as the object of our case study.

The AUV Under Test

Hardware Although we are only conducting experiments in simulators at this stage, our trained ML model can be easily deployed to real robots and the experiments are expected to be reproducible in real water tanks. Thus, we simulate the AUV in our laboratory—a customised BlueROV2, which has 4 vertical and 4 horizontal thrusters for 6 degrees of freedom motion. As shown in Figure 6.6-lhs, it is equipped with a custom underwater stereo camera designed for underwater inspection. A Water Linked A50 Doppler Velocity Log (DVL) is installed for velocity estimation and control. The AUV also carries an Inertial Measurement Unit (IMU), a depth sensor and a Tritech Micron sonar. The AUV is extended with an on-board Nvidia Jetson Xavier GPU computer and a Raspberry Pi 4 embedded computer. An external PC can also be used for data communication, remote control, mission monitoring, and data visualisation of the AUV via its tether.

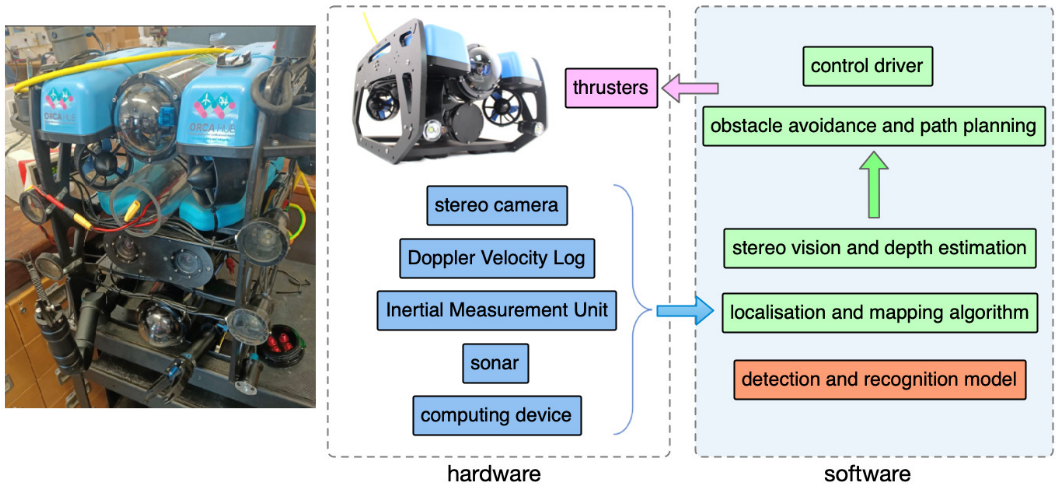

thrusters
stereo camera
Doppler Velocity Log
Inertial Measurement Unit
sonar
computing device
hardware
control driver
obstacle avoidance and path planning
stereo vision and depth estimation
localisation and mapping algorithm
detection and recognition model
software

Figure 6.6: Hardware–software architecture & key modules for autonomous survey & inspection missions.

Software Architecture With the hardware platform, we develop a software stack for underwater autonomy based on the Robot Operating System (ROS). The software modules that are relevant to the aforementioned AUV missions are (cf. Figure 6.6):

- Sensor drivers. All sensors are connected to on-board computers via cables, and their software drivers are deployed to capture real-time sensing data.
- Stereo vision and depth estimation. This is to process stereo images by removing its distortion and enhancing its image quality for inspection. After rectifying stereo images, they are used for estimating depth maps that are used for 3D mapping and obstacle avoidance.
- Localisation and mapping algorithm. In order to navigate autonomously and carry out a mission, we need to localise the vehicle and build a map for navigation. We develop a graph optimisation based underwater simultaneous localisation and mapping system by fusing stereo vision, DVL, and IMU. It also builds a dense 3D reconstruction model of structures for geometric inspection.
- Detection and recognition model. This is one of the core modules for underwater inspection based on ML models. It is designed to detect and recognise objects of interest in real-time. Based on the properties of detected objects— in particular the underwater assets to inspect—the AUV makes decisions on visual data collection and inspection.
- Obstacle avoidance and path planning. The built 3D map and its depth estimation are used for path planning, considering obstacles perceived by the stereo vision. Specifically, a local trajectory path and its way-points are generated in the 3D operating space based on the 3D map built from the localisation and mapping algorithm. Next the computed way-point is passed to the control driver for trajectory and way-point following.
- Control driver. We have a back seat driver for autonomous operations, enabling the robot to operate as an AUV. Once the planned path and/or a way-point is received, a Proportional-Integral-Derivative (PID) based controller is used to drive the thrusters following the path and approaching to the way-point. The controller can also be replaced by a learning based adaptive controller. While the robot moves in the environment, it continues perceiving the surrounding scene and processing the data using the previous software modules.

ML Model Doing Object Detection In this work, the state-of-the-art Yolo-v3 DL architecture [46] is used for object detection. Its computational efficiency and real-time performance are both critical for its application for underwater robots, as they mostly have limited on-board computing resources and power. The inference of Yolo can be up to 100 frames per second. Yolo models are also open source and built using the C language and the library is officially supported by OpenCV, which makes its integration with other AUV systems not covered in this work straightforward. Most DL-based object detection methods are extensions of a simple classification network. The object detection network usually generates a set of proposal bounding boxes; they might contain an object of interest and are then fed to a classification network. The Yolov3 network is similar in operation to, and is based on, the *darknet53* classification network.

The process of training the Yolo networks using the Darknet framework is similar to the training of most ML models, which includes data collection, model architecture implementation, and training. The framework consists of configuration files that can be set to match the number of object classes and other network parameters. Examples of training and testing data are described in Section 6.4.1 for simulated version of the model. The model training can be summarised by the following steps: i) define the number of object categories; ii) collect sufficient data samples for each category; iii) split the data into training and validation sets; and iv) use the Darknet software framework to train the model.

The Simulator

The simulator uses the popular Gazebo robotics simulator in combination with a simulator for underwater dynamics. The scenario models can be created/edited using Blender 3D software. We have designed the Ocean Systems Lab's wave tank model (cf. Figure 6.7-lhs) for the indoor simulated demo, using BlueROV2 within the simulation to test the scenarios. The wave tank model has the same dimension as our real tank. To ensure that the model

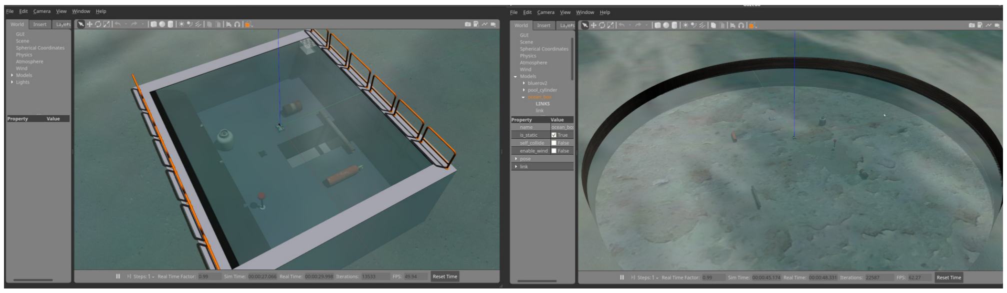


Figure 6.7: A wave-tank for simulated testing and a simulated pool for collecting the training data.

does not overfit the data, we have designed another scenario with a bigger pool for collecting the training data. The larger size allows for more distance between multiple objects, allowing both to broaden the set training scenarios and to make them more realistic. The simulated training environment is presented in Figure 6.7-rhs.

Our simulator creates configuration files to define an automated path using Cartesian way-points for the vehicle to follow autonomously, which can be visualised using Rviz. The pink trajectory is the desirable path and the red arrows represent the vehicle poses following the path, cf. Figure 6.8-lhs. There are six simulated objects in the water tank. They are a pipe, a gas tank, a gas canister, an oil barrel, a floating ball, and the docking cage, as shown in Figure 6.8-rhs. The underwater vehicle needs to accurately and timely detect them during the mission. Notably, the mission is also subject to random noise factors, so that repeated missions will generate different data that is processed by the learning-enabled components.

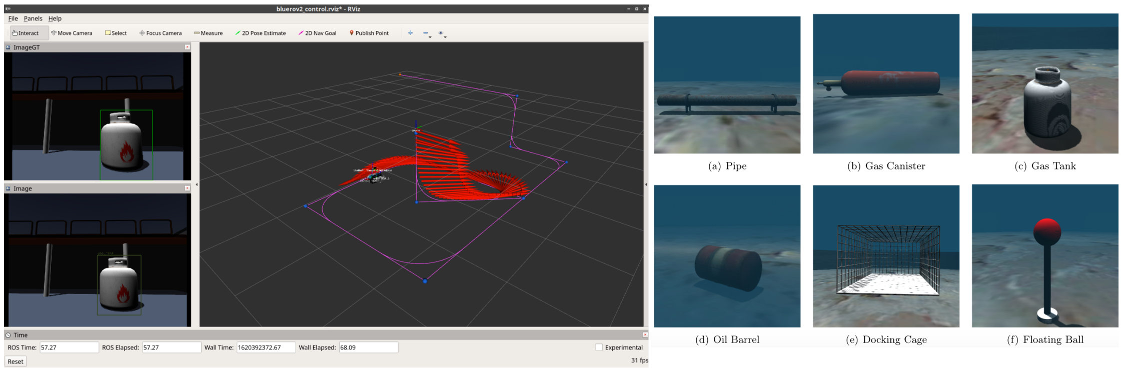


Figure 6.8: Simulated AUV missions following way-points and the six simulated objects.

6.4.2 Reliability Modelling of the AUV's Classification Function

Table 6.2: Average Precision (AP) of YOLOv3 for object detection.

Class	Train		Test	
	AP_{50}	AP_{75}	AP_{50}	AP_{75}
Pipe	0.98343	0.73503	0.97131	0.72532
Floating Ball	0.85765	0.40094	0.90912	0.42536
Gas Canister	0.87230	0.62546	0.87406	0.60331
Gas Tank	0.98930	0.76552	0.99346	0.76824
Oil Barrel	0.84578	0.61437	0.84258	0.57856
Docking Cage	0.88771	0.32021	0.91076	0.33656
mAP	0.90603	0.57692	0.91688	0.57289

Table 6.3: Reconstruction Loss & KL Divergence Loss of VAE model

VAE model	Train	Test
Recon. Loss	0.002601	0.003048
KL Div. Loss	1.732866	1.729756

Details of the YoloV3 model trained in this case study is presented in Table 6.2. We adopt the practical solutions discussed in Section 6.3.3 to deal with the high dimensionality of the collected operational dataset (256*256*3) by first training a VAE model and compressing the dataset into a new space with a much lower dimensionality of 8. While training details of the VAE model are summarised in Table 6.3, four sets of examples are shown in Figure 6.9,

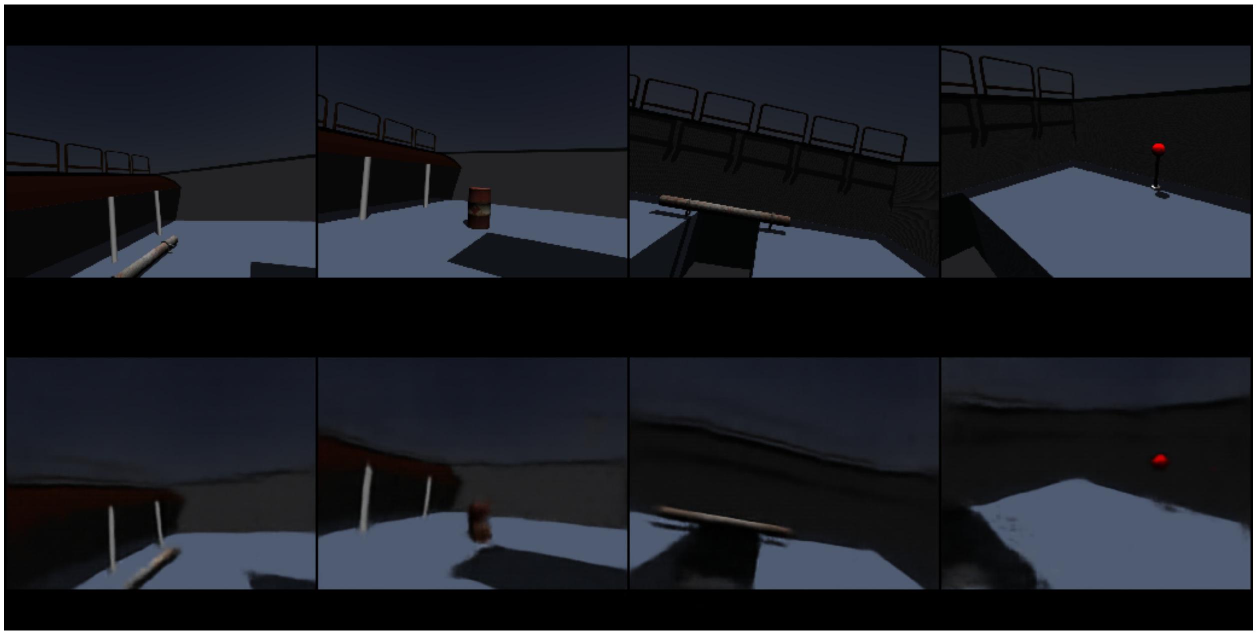


Figure 6.9: Four original images (top row) and the corresponding reconstructed images (bottom row) by the VAE model.

from which we can see that the reconstructed images are preserving the essential features of the objects (while blurring the less important background). We then choose a norm ball radius $\epsilon = 0.06$ according to the r-separation distance[15] and invoke the KDE and robustness estimator [115] for k randomly selected norm balls. Individual estimates of the k norm balls are then fed into the estimator for weighted average, Eqn.s (6.14) and (6.15). For comparison, we also calculate the ACU by assuming equal weights (i.e., a flat OP) in Eqn.s (6.14) and (6.15). Finally, the reliability claims on pmi and ACU are plotted as functions of k in Figure 6.10. Interpretation of the results is similar as before for CIFAR10, where the OP is also relatively flat. From the comparison results, it can be seen that the adversarial train can effectively improve the robustness of DL models and be captured by our RAM method.

RAM is scalable to high dimensional data for assessing the reliability of YoloV3 model, the real world object detection CNNs.

[15]Because more than one object may appear in a single image, the label of the "dominating" object (e.g., the object with the largest bounding box and/or with higher priority) can be used in the calculation of r. For simplicity, we first preprocess the dataset by filtering out images with multiple labels, and then determine the ϵ based on an estimated r.

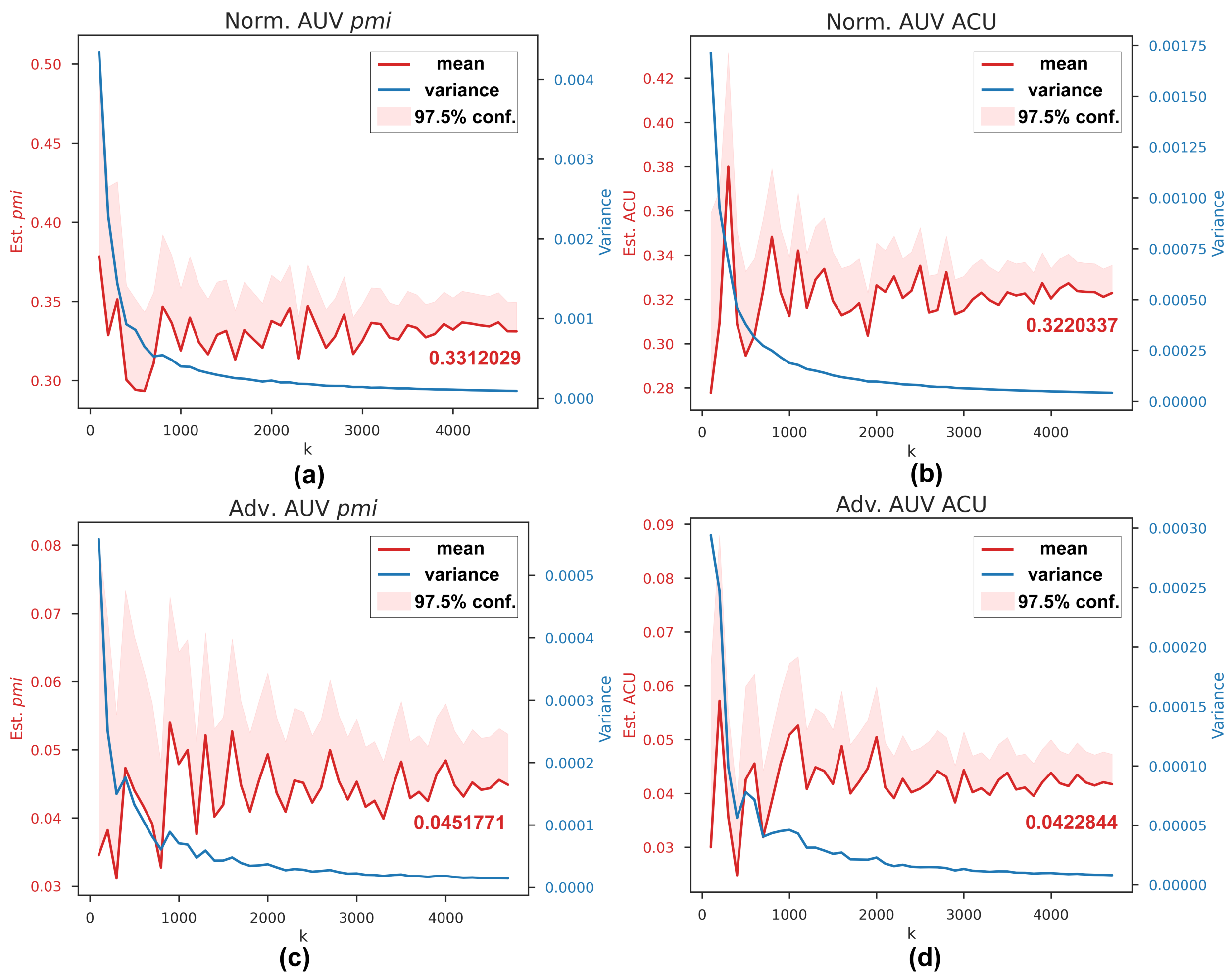


Figure 6.10: The mean, variance and 97.5% confidence upper bound of AUV's *pmi* and ACU as functions of k sampled norm balls.

6.5 Discussions on the Proposed RAM

In this section, we summarise the *model assumptions* made in our RAM, and discuss if/how they can be validated and which new assumptions and compromises in the solutions are needed to cope with real-world applications with high-dimensional data. Finally, we list the *inherent difficulties* of assessing ML reliability uncovered by our RAM.

R-Separation and its Estimation Assumption 1 derives from Remark 12. We concur with [123] and believe that, for any real-world ML classification application where the inputs are data-points with "physical meanings", there should always exist an r-stable ground truth. Such r-stable ground truth varies between applications, and the smaller the r is, the harder the inherent difficulty of the classification problem becomes. This r is therefore a *difficulty indicator* for the given classification problem. Indeed, it is hard to estimate the r (either in the input pixel space nor the latent feature space)—the best we can do is to estimate it from the existing dataset. One way of solving the problem is to keep monitoring the r estimates as more labelled data is collected, e.g. during operation, and to redo the cell partition when the estimated r has changed significantly. Such a dynamic way of estimating r can be supported by the concept of dynamic assurance cases [173].

Approximation of the OP from Data Assumption 2 says that the collected dataset statistically represents the OP, which may not hold for many practical reasons—e.g., when the future OP is uncertain at the training stage and data is therefore collected in a balanced way to perform well in all categories of inputs. Although we demonstrate our RAM under this assumption for simplicity, it can be easily relaxed. Essentially, we try to fit a PDF over the input space from an "operational dataset" (representing the OP). Data-points in this set can be *unlabelled* raw data generated from historical data of previous applications and simulations, which can then be scaled based on domain expert knowledge (e.g., by DL generative models that we are currently investigating). Obtaining such an operational dataset is an application-specific engineering problem, and manageable thanks to the fact that it does not require labelled data. Notably, the OP may also be approximated at *runtime* based on the data stream of operational data. Efficient KDE for data streams [174] can be used. If the OP was subject to sudden changes, change-point detectors like [175] should also be paired with the runtime estimator to robustly approximate the OP. Again, such dynamic way of estimating OP can also be supported by dynamic assurance cases [173].

Determination of the Ground Truth of a Cell Assumptions 3 and 4 are essentially on how to determine the ground truth label for a given cell, which relates to the oracle problem of testing ML software. While this still remains challenging, we partially solve it by leveraging the r-separation property. Thanks to r, it is easy to determine a cell's ground truth when we see that it contains labelled data-points. However, for an empty cell, it is non-trivial. We assume the overall performance of the ML model is fairly good (e.g., better than a classifier doing random classifications), thus misclassifications within an empty cell are relatively rare events. We can determine the ground truth label of the cell by majority voting of predictions. Indeed, it is a strong assumption when there are some "failure regions" in the input space, within which the ML model performs really badly (even worse than random labelling). In this case, we need new mechanism to detect such "really bad failure regions" or spend more budget on, for example, asking humans to do the labelling.

Efficiency of Cell Robustness Evaluation Although we only applied the two methods of SMC and [115] in our experiments to evaluate the local robustness, we believe that other statistical sampling methods designed for estimating the probability of rare-events could be used as well. Moreover, the cell robustness estimator in our RAM works in a "hot-swappable" manner: any new and more efficient estimator can easily be incorporated. Thus, despite being an important question, how to improve the efficiency of the robustness estimation for cells is beyond the scope of our RAM.

Conditional OP of a Cell We assume that the distribution of inputs (the conditional OP) within each cell is uniform by Assumption 5. Although we conjecture that this is the common case due to the small size of cells (i.e., those very close/similar inputs within a small region are only subject to noise factors that can be modelled uniformly), the specific situation may vary; this requires justification in safety cases. For a real-world dataset, the conditional OP might represent certain distributions of "natural variations" [176], e.g. lighting conditions, that obey certain distributions. Ideally, the conditional OP of cells should capture the distribution of such natural variations. Recent advance on measuring the naturalness/realisticness of AEs [31] relates to this assumption and may relax it.

Independent λ_is and Op_is As per Assumption 6, we assume all λ_is and Op_is are independent when "assembling" their estimates via Eqn. (6.11) and deriving the variance via Eqn. (6.12). This assumption is largely for the mathematical tractability when propagating the confidence in individual estimates at the cell-level to the *pmi*. Although this independence assumption is hard to justify in practice, it is not unusual in reliability models that do partition, e.g., in [164, 177]. We believe that RAMs are still useful under this assumption, while we envisage that Bayesian estimators leveraging joint priors and conjugacy may relax it.

Uncertainties Raised by Individual OP and Robustness Estimates This relates to how reliable the chosen OP and robustness estimators themselves are. Our RAM is flexible and evolvable in the sense that it does not depend on any specific estimators. New and more reliable estimators can therefore easily be integrated to reduce the estimation uncertainties. Moreover, such uncertainties raised by estimators are propagated and compounded in our overall RAM results, cf. Eqn.s (6.12) and (6.15). Although we ignore them as per Assumption 7, this is arguably the case when the two estimators are fairly reliable and the number of samples k is much smaller than the sample frame size n.

Inherent Difficulties of Reliability Assessment on ML Software Finally, based on our RAM and the discussions above, we summarise the inherent difficulties of assessing ML reliability as the following questions:

- How to accurately learn the OP in a potentially high-dimensional input space with relatively sparse data?
- How to build an accurate test oracle (to determine the ground truth label) by, e.g., leveraging the existing labels (done by humans) in the training dataset?
- What is the local distribution (i.e. the conditional OP) over a small input region (which is potentially only subject to subtle natural variations of physical conditions in the environment)?

- How to efficiently evaluate the robustness of a small region, given that AEs are normally rare events? And how to reduce the risk associated with an AE (e.g., referring to ALARP)?

- How to efficiently sample small regions from a large population (due to the high-dimensionality) of regions to test the local robustness in an unbiased and uncertainty informed way, given a limited budget?

We provide solutions in our RAM that are practical compromises (cf. Section 6.3.3), while the questions above are still challenging and generic. For some domains, say self-driving cars, they are relatively easier to tackle, thanks to the large amount of available data—not only historical data, but also millions of miles of public road testing data are collected in recent years. At this stage, we doubt the existence of other RAMs for ML software with weaker assumptions that achieve the same level of rigorousness as ours, in which sense our RAM advances in this research direction.

Chapter 7

Practical Verification of DL Safety in State Estimation Systems

7.1 Introduction

State estimation systems have been widely deployed for many tasks within robotic applications, including localisation [178], tracking [179], and control [180]. This paper considers neural network enabled state estimation systems where neural networks are used to process perceptional input received via sensors. More and more robotics applications adopt neural network components to take advantage of their high prediction precision [181].

However, the neural network has been found to be vulnerable to adversarial attacks, i.e., a small, imperceptible perturbation on the input may alter the classification output. The concern has been raised on how safe a learning enabled system is when learning components interact with other components (including Bayes filter components). In [182], it has been found that the system is able to compensate (to some degree) against adversarial attacks on its neural network component, but there may also be new uncertainties from the interactions between learning and non-learning components. While *some* vulnerability cases were reported in [182], there is no comprehensive study on *all* potential risks in a learning enabled system, from the perspective of formal verification. Formal verification can prove that a system is correct against *all* possible risks over a specification and the formal model of the system, and returns counterexamples when it cannot. The ability to sufficiently identify risks is necessary for the deployment of safety critical applications – this chapter addresses this need. It is the *first time a verification approach has been developed for securing learning enabled state estimation systems.*

Technically, we first formalise a state estimation system as a novel labelled transition system which has components for payoffs and partial order relations. Specifically, every transition is attached with a payoff, and for every state there is a partial order relation between its out-going transitions from the same state. Second, we show that the verification of the robustness property on such a system can be reduced into a constrained optimisation problem. Third, to enable practical verification, we develop two algorithms: (1) a verification algorithm – that can achieve complete results but cannot be used for run-time – and (2) a heuristic algorithm – that can be used efficiently in run-time, perform well in most cases, but cannot provide a completeness guarantee.

As a major case study, we work with a real-world dynamic tracking system [180], which detects and tracks ground vehicles over the high-resolution Wide Area Motion Imagery (WAMI) data stream, named *WAMI tracking system* in this paper. We apply the developed algorithms to the WAMI tracking system for safety analysis, in particular, we consider on a robustness property that concerns whether the system can function well under attack on the perceptional neural network components.

7.2 Preliminaries

7.2.1 Neural Networks

Let $\mathcal{X}$ be the input domain and $\mathcal{Y}$ be the set of labels. A neural network $\mathcal{N} : \mathcal{X} \to \mathcal{D}(\mathcal{Y})$ can be seen as a function mapping from $\mathcal{X}$ to probabilistic distributions over $\mathcal{Y}$. That is, $\mathcal{N}(x)$ is a probabilistic distribution, which assigns for each label $y \in \mathcal{Y}$ a probability value $(\mathcal{N}(x))_y$. We let $f_{\mathcal{N}} : \mathcal{X} \to \mathcal{Y}$ be a function such that for any $x \in \mathcal{X}$, $f_{\mathcal{N}}(x) = \arg\max_{y\in\mathcal{Y}}\{(\mathcal{N}(x))_y\}$, i.e., $f_{\mathcal{N}}(x)$ returns the classification.

7.2.2 Neural Network Enabled State Estimation

We consider a time-series linear state estimation problem that is widely assumed in the context of object tracking. The process model is defined as follow.

$$\mathbf{s}_k = \mathbf{F} \cdot \mathbf{s}_{k-1} + \omega_k \tag{7.1}$$

where $\mathbf{s}_k$ is the state at time k, $\mathbf{F}$ is the transition matrix, ω_k is a zero-mean Gaussian noise such that $\omega_k \sim \mathcal{N}(0, \mathbf{Q})$, with $\mathbf{Q}$ being the covariance of the process noise. Usually, the states are not observable and need to be determined indirectly by measurement and reasoning. The measurement model is:

$$\mathbf{z}_k = \mathbf{H} \cdot \mathbf{s}_k + \mathbf{v}_k \tag{7.2}$$

where $\mathbf{z}_k$ is the observation, $\mathbf{H}$ is the measurement matrix, $\mathbf{v}_k$ is a zero-mean Gaussian noise such that $\mathbf{v}_k \sim \mathcal{N}(0, \mathbf{R})$, and $\mathbf{R}$ is the covariance of the measurement noise.

Bayes filters have been used for reasoning about the observations, $\{\mathbf{z}_k\}$, with the goal of learning the underlying states $\{\mathbf{s}_k\}$. A Bayes filter maintains a pair of variables, $(\mathbf{s}_k, \mathbf{P}_k)$, over the time, denoting Gaussian estimate and uncertainty, respectively. The basic procedure of a Bayes filter is to use a transition matrix, $\mathbf{F}_k$, to predict the current state, $(\hat{\mathbf{s}}_k, \hat{\mathbf{P}}_k)$, given the previous state, $(\mathbf{s}_{k-1}, \mathbf{P}_{k-1})$. The prediction state can be updated into $(\mathbf{s}_k, \mathbf{P}_k)$ if a new observation, $\mathbf{z}_k$, is obtained. In the context of the aforementioned problem, this procedure is iterated for a number of time steps, and is always discrete-time, linear, but subject to noises.

We take the Kalman Filter (KF), one of the most widely used variants of Bayes filter, as an example to demonstrate the above procedure. Let $\mathbf{s}_0 \in \Re^n \sim \mathcal{N}(\hat{\mathbf{s}}_0, \hat{\mathbf{P}}_0)$ be the initial state, such that $\hat{\mathbf{s}}_0 \in \Re^n$ and $\hat{\mathbf{P}}_0 \in \Re^{n\times n}$ represent our knowledge about the initial estimate and the corresponding covariance matrix, respectively.

First, we perform the **state prediction** for $k \geq 1$:

$$\begin{array}{rcl} \hat{\mathbf{s}}_k & = & \mathbf{F}_k \mathbf{s}_{k-1} \\ \hat{\mathbf{P}}_k & = & \mathbf{F}_k \mathbf{P}_{k-1} \mathbf{F}_k^T + \mathbf{Q}_k \end{array} \tag{7.3}$$

Then, we can **update the filter**:

$$\begin{array}{rcl} \mathbf{s}_k & = & \hat{\mathbf{s}}_k + \mathbf{K}_k \mathbf{y}_k \\ \mathbf{P}_k & = & (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \hat{\mathbf{P}}_k \end{array} \tag{7.4}$$

such that

$$\begin{array}{rcl} \mathbf{y}_k & = & \mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{s}}_k \\ \mathbf{S}_k & = & \mathbf{H}_k \hat{\mathbf{P}}_k \mathbf{H}_k^T + \mathbf{R}_k \\ \mathbf{K}_k & = & \hat{\mathbf{P}}_k \mathbf{H}_k^T \mathbf{S}_k^{-1} \end{array} \tag{7.5}$$

Intuitively, $\mathbf{y}_k$ is usually called "innovation" in signal processing that represents the difference between the real observation and the predicted observation, $\mathbf{S}_k$ is the covariance matrix of this innovation, and $\mathbf{K}_k$ is the Kalman gain, representing the relative importance of innovation $\mathbf{y}_k$ with respect to the predicted estimate $\hat{\mathbf{s}}_k$.

In a neural network enabled state estimation, a perception system – which may include multiple CNNs – will provide a set of candidate observations Z_k, any of which can be chosen as the new observation $\mathbf{z}_k$. From the perspective of robotics, Z_k includes a set of possible states of the robot, measured by (possibly several different) sensors at time k. These measurements are imprecise, and are subject to noises from both the environment (called epistemic uncertainty) and the imprecision of sensors (aleatory uncertainty).

7.2.3 A Real-World WAMI Dynamic Tracking System

In this part, we have a brief introduction to the real-world WAMI dynamic tracking system that will be used as our major case study. The details of this system can be found in [182] and [180]. The tracking system requires continuous imagery input from e.g., airborne high-resolution cameras. The input is a video, which consists of a finite sequence of WAMI images. Each image contains a number of vehicles. The processing chain of the WAMI tracking system is as follows.

1. Align a set of previous frames with the incoming one.

2. Construct the background model of incoming frames using the median frame.

3. Extract moving objects using background subtraction.

4. Determine if the moving objects are vehicles by using a Binary convolutional neural networks (CNN).

5. For complex cases, predict the locations of moving objects/vehicles using a regression CNN.

6. Track one of the vehicles using a Kalman filter.

WAMI tracking uses **Gated Nearest Neighbour (Gnn)** to choose the new observation $\mathbf{z}_k$: from the set Z_k, the one closest to the predicted measurement $\mathbf{H}_k \cdot \hat{\mathbf{s}}_k$ is chosen, i.e.,

$$\mathbf{z}_k = \arg\min_{\mathbf{z} \in Z_k} ||\mathbf{z} - \mathbf{H}_k \cdot \hat{\mathbf{s}}_k||_p \tag{7.6}$$

$$s.t. \quad ||\mathbf{z} - \mathbf{H}_k \cdot \hat{\mathbf{s}}_k||_p \leq \epsilon_k \tag{7.7}$$

where $|| \cdot ||_p$ is for the L^p-norm distance (p=2, i.e., Euclidean distance, is used in this paper), and ϵ_k is the gate value, representing the maximum uncertainty the system is able to work with.

Specifically, the WAMI system has the following $\mathbf{s}$ and $\mathbf{P}$:

$$\mathbf{s} \;=\; \begin{bmatrix} l \\ v \end{bmatrix} \qquad \mathbf{P} \;=\; \begin{bmatrix} \Sigma_{ll} & \Sigma_{lv} \\ \Sigma_{vl} & \Sigma_{vv} \end{bmatrix} \tag{7.8}$$

where $\mathbf{s}$ contains the currently best estimates – or mean values – of two variables l, representing the location, and v, representing the velocity, respectively. Elements of $\mathbf{P}$, Σ_{ij}, represent the degrees of correlation between variables $i, j \in \{l, v\}$. The diagonal of $\mathbf{P}$ contains the mean square error of the estimate $\mathbf{s}$. The **uncertainty – or Bayesian uncertainty –** ϵ is the trace of the covariance matrix:

$$\epsilon = tr(\mathbf{P}) = \Sigma_{ll} + \Sigma_{vv} \tag{7.9}$$

Intuitively, ϵ denotes a search range and only within this range, observations are considered. Normally, ϵ will gradually shrink before being bounded – a convergence property of the KF as explained below.

We remark that this WAMI dynamic tracking system models a Poisson-Bernoulli mixture process. The track initialisation and the occurrence of false alarms follow a Poisson distribution (since we are focusing on one single target, we choose not to explicitly model this part) and the detectability of the target follows a Bernoulli distribution which models the CNN measurements (including mis-detections and spatial errors). In this paper, KF is used to address the measurement noise, but we note that it cannot be ensured that the Gaussian noise is sufficient in this case. Therefore, the usage of KF here is only under the assumption of Gaussian noises whose parameters are empirically configured. It can be seen as a smoothing algorithm rather than an optimal solution. Nevertheless, investigating alternative likelihoods that more faithfully represent the uncertainty and automated parameter estimation for this problem is an intriguing future work.

7.2.4 Expansion of Bayesian Uncertainty in Kalman Filters

Generally, a KF system works with the two phases, prediction (7.3) and update (7.4), alternating. Theoretically, KFs converge [183] under a good set of parameters $\mathbf{F}$, $\mathbf{H}$, $\mathbf{Q}$, and $\mathbf{R}$. In this paper, we assume that the KF system has been well designed to ensure the convergence. Empirically, this has been proven possible in many practical systems. We are interested in another property of KF, i.e., the uncertainty in $\hat{\mathbf{P}}_k$ increases as opposed to $\mathbf{P}_k$; if no update phase, i.e., Equation (7.4), is held, this predicted covariance $\hat{\mathbf{P}}_k$ will be carried over to the next step as $\mathbf{P}_{k+1}$. In the WAMI tracking system, when observation $\mathbf{z}_k$ is unavailable within ϵ for some reason, the update step can be skipped and multiple prediction steps are performed consecutively. In this case, the Bayesian uncertainty ϵ may 'explode', and the search range of observations is expanded. We will explain later this property can be utilised to design a monitor to counter the attack, and therefore should be considered when analysing the robustness.

7.3 Problem Formulation

7.3.1 Threat Model of Adversarial Attack on Perception System

In Section 7.2.3, a neural network based perception system determines whether or not there is a vehicle at a location $\mathbf{z}$. Let $x(\mathbf{z}) \in \Re^{d_1 \times d_2}$ be an image covering the location $\mathbf{z}$, a neural network function $f_{\mathcal{N}} : \Re^{d_1 \times d_2} \to \{0, 1\}$ maps $x(\mathbf{z})$ into a Boolean value $f_{\mathcal{N}}(x(\mathbf{z}))$ representing whether or not a vehicle is present at location $\mathbf{z}$. There are two types of erroneous detection: (1) a wrong classification prediction of the image $x(\mathbf{z})$, and (2) a wrong positioning of a moving object within $x(\mathbf{z})$. We focus on the former since the WAMI tracking system has a comprehensive mechanism to prevent the occurrence of the latter.

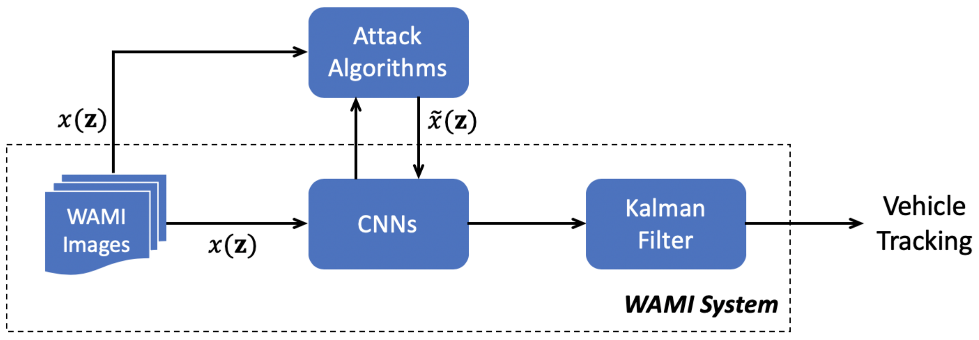


Figure 7.1: The workflow of attacking the WAMI system.

The threat model of an adversary is depicted as in Figure 7.1. Assuming that $f_{\mathcal{N}}(x(\mathbf{z})) = 1$, an adversary is to compute another input $\widetilde{x}(\mathbf{z})$ with a certain payoff to have a different classification, i.e., $f_{\mathcal{N}}(\widetilde{x}(\mathbf{z})) = 0$. Without loss of generality, the *payoff* is measured with the norm-distance from $\widetilde{x}(\mathbf{z})$ to its original image $x(\mathbf{z})$, or formally

$$||\widetilde{x}(\mathbf{z}) - x(\mathbf{z})||_p \tag{7.10}$$

To deviate from an input image $x(\mathbf{z})$ to its adversarial input $\widetilde{x}(\mathbf{z})$, a large body of adversarial example generation algorithms and adversarial test case generation algorithms are available [5, 184]. Formal verification based methods such as [185, 186, 187] can also be used. Given a neural network $\mathcal{N}$ and an input x, an adversarial algorithm A produces an adversarial example $A(\mathcal{N}, x)$ such that $f_{\mathcal{N}}(A(\mathcal{N}, x)) \neq f_{\mathcal{N}}(x)$. On the other hand, for test case generation, an algorithm A produces a set of test cases $A(\mathcal{N}, x)$, among which the optimal adversarial test case is such that $\arg\min_{\widetilde{x} \in A(\mathcal{N},x), f_{\mathcal{N}}(\widetilde{x}) \neq f_{\mathcal{N}}(x)} ||\widetilde{x} - x||_p$. We remark that, the work in this paper is independent from particular adversarial algorithms. We use in our experiments two algorithms:

- DeepFool [188], which finds an adversarial example $\widetilde{x}$ by projecting x onto the nearest decision boundary.
- DeepConcolic [51], which generates test cases by applying combined symbolic and concrete execution, guided by adapted MC/DC metrics for neural networks.

We denote by $\mathit{payoff}(A, \mathcal{N}, x)$, the payoff that an algorithm A needs to compute an adversarial example from x and $\mathcal{N}$. Furthermore, we assume that the adversary can observe the parameters of the Bayes filter, for example, $\mathbf{H}_k, \mathbf{F}_k$, $\mathbf{Q}_k, \mathbf{R}_k$ of the Kalman filter.

7.3.2 $\{\text{PO}\}^2$Labelled Transition Systems

Let Prop be a set of atomic propositions. A payoff and partially-ordered label transition system, or $\{\text{PO}\}^2$-LTS, is a tuple $M = (Q, q_0, \mathit{kf}, L, \alpha, \beta)$, where Q is a set of states, $q_0 \in Q$ is an initial state, $\mathit{kf} \subseteq Q \times Q$ is a transition relation, $L : Q \to 2^{\mathit{Prop}}$ is a labelling function, $\alpha : Q \times Q \to \Re^+$ is a payoff function assigning every transition a non-negative real number, and $\beta : \mathit{kf} \to \mathit{kf}$ is a partial order relation between out-going transitions from the same state.

7.3.3 Reduction of WAMI Tracking to $\{\text{PO}\}^2$LTS

We model a neural network enabled state estimation system as a $\{\text{PO}\}^2$-LTS. A brief summary of some key notations in this paper are in Table 7.1. We let each pair $(\mathbf{s}_k, \mathbf{P}_k)$ be a state, and use the transition relation kf to model the transformation from a pair to another pair in a Bayes filter. We have the initial state q_0 by choosing a detected vehicle $(\mathbf{s}_0, \mathbf{P}_0)$ on the map. From a state $q_{k-1} = (\mathbf{s}_{k-1}, \mathbf{P}_{k-1})$ and a set Z_k of candidate observations, we have one transition (q_{k-1}, q_k) for each $\mathbf{z} \in Z_k$, where $q_k = (\mathbf{s}_k, \mathbf{P}_k)$ can be computed with Equations (7.3)-(7.5) by having $\mathbf{z}_k$ in Equation (7.6) as the new observation. For a state $q_k = (\mathbf{s}_k, \mathbf{P}_k)$, we write $\mathbf{s}(q_k)$ to denote the estimate $\mathbf{s}_k$, $\mathbf{P}(q_k)$ to denote the covariance matrix

Notations	Description
$\mathbf{z}_k \in Z_k$	observed location by WAMI tracking
$x(\mathbf{z})$	an $d_1 \times d_2$ image covering location $\mathbf{z}$
f_N	neural network function
$payoff(A, \mathcal{N}, x)$	payoff for algorithm A computing an adversarial example from x and $\mathcal{N}$
$q_k = (\mathbf{s}_k, \mathbf{P}_k)$	a state at step k, consisting of estimate and covariance matrix
$\mathbf{s}(q_k)$, $\mathbf{P}(q_k)$ and $\mathbf{z}(q_k)$	estimate of q_k, covariance matrix of q_k and observed location for transition (q_{k-1}, q_k)
ρ	a path of consecutive states $q_l ... q_u$

Table 7.1: A Summary of Notations Used

$\mathbf{P}_k$, and $\mathbf{z}(q_k)$ to denote the new observation that has been used to compute $\mathbf{s}(q_k)$ and $\mathbf{P}(q_k)$ from its parent state q_{k-1}.

Subsequently, for each transition (q_{k-1}, q_k), its associated payoff $\alpha(q_{k-1}, q_k)$ is denoted by $payoff(A, \mathcal{N}, x(\mathbf{z}(q_k)))$, i.e., the payoff that the adversary uses the algorithm A to manipulate $x(\mathbf{z}(q_k))$ – the image covering the observation $\mathbf{z}(q_k)$ – into another image on which the neural network $\mathcal{N}$ believes there exists no vehicle.

For two transitions (q_{k-1}, q_k^1) and (q_{k-1}, q_k^2) from the same state q_{k-1}, we say that they have a partial order relation, written as $(q_{k-1}, q_k^1) \prec (q_{k-1}, q_k^2)$, if making $\mathbf{z}(q_k^2)$ the new observation requires the adversary to fool the network $\mathcal{N}$ into misclassifying $x(\mathbf{z}(q_k^1))$. For example, in WAMI tracking, according to Equation (7.6), the condition means that $||\mathbf{z}(q_k^2) - \mathbf{z}||_p > ||\mathbf{z}(q_k^1) - \mathbf{z}||_p$, where $\mathbf{z} = \mathbf{H}_k \mathbf{s}(q_{k-1})$ is the predicted location.

Example 7.3.1. *Figure 7.2 depicts a tree diagram for the unfolding of a labelled transition system. The root node on top represents the initial state q_0. Each layer comprises all possible states of $q_k = (\mathbf{s}_k, \mathbf{P}_k)$ at step k of WAMI tracking, with $\mathbf{s}_k$ being one possible estimate, and $\mathbf{P}_k$ the covariance matrix. Each transition connects a state q_{k-1} at step $k-1$ to q_k at step k. $\ldots, \mathbf{z}_{k-1}, \mathbf{z}_k, \mathbf{z}_{k+1}, \mathbf{z}_{k+2}, \ldots$ are the observed locations at each step by WAMI tracking.*

Given a $\{\text{PO}\}^2$-LTS M, we define a path ρ as a sequence of consecutive states $q_l ... q_u$, and $\mathbf{z}(\rho)$ as a sequence of corresponding observed location $\mathbf{z}_l ... \mathbf{z}_u$ for $0 \leq l < u$, where l and u are the starting and ending time under consideration, respectively. We write ρ_k for the state q_k, and $\mathbf{z}(\rho_k)$ for the observed location $\mathbf{z}(q_k)$ on the path ρ.

7.3.4 A Simple Monitor on Bayesian Uncertainty

Given the convergence of the KF (as explained in Section 7.2.4), we can easily design a system to monitor the Bayesian uncertainty: whenever there is an increase of the uncertainty range

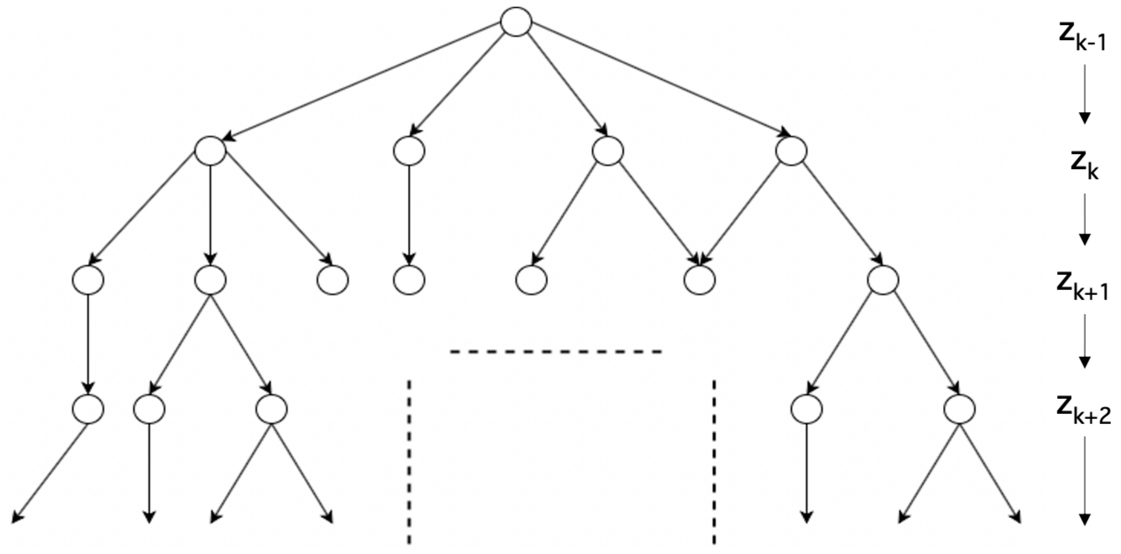


Figure 7.2: Tree diagram of an unfolding $\{\text{PO}\}^2$-LTS

ϵ, an alarm is set to notify the potential attack. To overcome this monitor, we require that a successful attack should not present an increase on ϵ. To understand when the increase may appear for the WAMI tracking system, we recall the discussion in Section 7.2.3 that a tracking associates the nearest observation $\mathbf{z}$ within the uncertainty range ϵ at each step. When no observations are available in the range, e.g., all the observations in Z_k are attacked, only Equation (7.3) is performed and Equation (7.4) is skipped. In this case, the Bayesian uncertainty increases due to the noise $\mathbf{Q}_k$ and the transition matrix $\mathbf{F}_k$.

7.3.5 Specification as Optimization

Verification determines whether a specification ϕ holds on a given LTS M [92]. Usually, a logic language, such as CTL, LTL, or PCTL, is used to formalize the specification ϕ. In this paper, to suit our needs, we let the specification ϕ be a constrained optimisation objective, and then the verification is to determine whether, given M and ϕ, there is a solution to the optimisation problem. If the answer is affirmative, an optimal solution is returned.

We focus on the specification that expresses the robustness of the system – ***given an adversary, whether or not the state estimation system is able to function well with only minor loss of localisation precision?*** – but remark that the verification algorithm can be extended to work with other specifications.

First, we consider the measure for the loss of localisation precision. Let ρ be an original path that has not suffered an attack. We define $dist(\rho, \widetilde{\rho})$ as the distance between ρ and the other path $\widetilde{\rho}$, which is obtained after an attack, and say that the system is robust to the attack if $dist(\rho, \widetilde{\rho}) < \theta_{robustness}$ for a threshold $\theta_{robustness} > 0$. For the WAMI tracking system, we define

$$dist(\rho, \widetilde{\rho}) = (\sum_{k=l}^{u} ||\mathbf{z}(\rho_k) - \mathbf{z}(\widetilde{\rho}_k)||_p)/(u - l + 1) \tag{7.11}$$

as the averaged norm distance between two given times l and u. It is straightforward to see that, if we can find the maximally allowed distance $\max_{\widetilde{\rho}} dist(\rho, \widetilde{\rho})$ then we can firmly conclude – through verification – whether or not the system is robust on the path ρ, by comparing a given distance d with $\max_{\widetilde{\rho}} dist(\rho, \widetilde{\rho})$.

In addition to the satisfaction of the objective as above, we require the *best* attack to not be easily detected by e.g., monitors. In this paper, we consider two monitors who look after two quantities, as explained below. Firstly, we consider a monitor Γ which looks after the Kalman filter's state by considering its convergence. Γ has the following definition:

$$\Gamma(\widetilde{\rho}_{k-1}, \widetilde{\rho}_k) = \begin{cases} 1 & \epsilon(\widetilde{\rho}_k) \leq \epsilon(\widetilde{\rho}_{k-1}) \\ 0 & \epsilon(\widetilde{\rho}_k) > \epsilon(\widetilde{\rho}_{k-1}) \end{cases} \tag{7.12}$$

Basically, Γ continuously checks the value of uncertainty ϵ, which is derived from covariance $\mathbf{P}$. It is to capture the case where the uncertainty increases, as required by Section 7.3.4.

We consider the other simple monitor which looks after the (mean) payoff in attacking the perception system and, once the payoff is over a threshold θ_{payoff} the attacker is detected. Let $(\widetilde{\rho}_{k-1}, \widetilde{\rho}_k)$ be a transition on an attack path $\widetilde{\rho}$, we have

$$\varphi(\widetilde{\rho}_{k-1}, \widetilde{\rho}_k) = \sum_{(\widetilde{\rho}_{k-1}, \widetilde{\rho}_k^{\diamond}) \prec (\widetilde{\rho}_{k-1}, \widetilde{\rho}_k)} \alpha(\widetilde{\rho}_{k-1}, \widetilde{\rho}_k^{\diamond}) \tag{7.13}$$

as the *combined payoffs* that are required to implement the transition $(\widetilde{\rho}_{k-1}, \widetilde{\rho}_k)$. Intuitively, all the payoffs of the transitions $(\widetilde{\rho}_{k-1}, \widetilde{\rho}_k^{\diamond})$, which are partially ordered by the envisaged transition $(\widetilde{\rho}_{k-1}, \widetilde{\rho}_k)$, are counted. In the WAMI tracking system, this means that the attack results in misclassifications of all the images $x(\mathbf{z}(\widetilde{\rho}_k^{\diamond}))$ with $\mathbf{z}(\widetilde{\rho}_k^{\diamond})$ being closer to the predicted location $\mathbf{F}_k\mathbf{s}(\widetilde{q}_{k-1})$ than $\mathbf{z}(\widetilde{\rho}_k)$.

Therefore, the optimisation problem can be formulated as

$$\begin{aligned} \underset{\widetilde{\rho}}{\textbf{maximize}} \quad & dist(\widetilde{\rho}, \rho) \\ \textbf{subject to} \quad & \sum_{k=l+1}^{u} \Gamma(\widetilde{\rho}_{k-1}, \widetilde{\rho}_k) = u - l \\ & \sum_{k=l+1}^{u} \varphi(\widetilde{\rho}_{k-1}, \widetilde{\rho}_k) \leq (u - l) \cdot \theta_{payoff} \end{aligned} \tag{7.14}$$

where $dist(\widetilde{\rho}, \rho)$ is the deviation from the original track ρ to the adversarial track $\widetilde{\rho}$, $\sum_{k=l+1}^{u} \Gamma(\widetilde{\rho}_{k-1}, \widetilde{\rho}_k)$ is the monitoring of convergence of tracking to enable $\widetilde{\rho}$, and

$$\varepsilon_{avg} = \sum_{k=l+1}^{u} \varphi(\widetilde{\rho}_{k-1}, \widetilde{\rho}_k)/(u - l) \tag{7.15}$$

is the mean payoff from time l to u. Finally, the verification problem is to compute the above optimisation objective on a given $\{\text{PO}\}^2$-LTS M.

7.4 Automated Verification

An attack on the WAMI system, as in Section 7.3.1, adds perturbations to the images containing vehicles in order to fool the neural network into making a wrong detection. Then, this wrong detection will be passed on to the KF-based tracking system. For the KF, a detection is adopted as an observation w.r.t. the Gnn, as explained in Section 7.2.3. In this section, we develop algorithms to find the maximally deviated path, which will be compared with a given threshold $\theta_{robustness}$ to have the verification result.

7.4.1 Baseline Method

We consider a baseline method that does not take into account the monitor on Bayesian uncertainty (Equation (7.11)) or the other monitor on attack payoff (Equation (7.13)); it simply attacks the neural networks to make the currently associated observation – the nearest one to the prediction – unseen to the KF. This is the method used in [182].

7.4.2 Verification Algorithm based on Exhaustive Search

Our verification algorithm proceeds by exhaustively computing over all possible attacked tracks. Since a final deviation is not available until the end of a simulation, the tree has to be fully expanded from the root to the leaf and all the paths are explored. Breadth-first search (BFS) is used to find the best solution.

The details are in Algorithm 6. We have several operation functions on the tree diagram for the labelled transition system. $leaf$ returns all leaf nodes of the given root node. $parent$ associates a node to its parent node. $path$ returns all tree paths from the given root node to the leaf nodes.

Lines 2-15 present the procedure of constructing the tree diagram. First, we set the root node ρ_l (Line 2), that is, we will attack the system from the (l)-th state of the original path ρ and enumerate all possible adversarial tracks. At each step k, function $neighbours$ will list all observations near the predicted location (Line 5). Then, each observation is incorporated with current state ρ_k for the calculation of next state ρ_{k+1} (Line 7). To enable each transition (ρ_k, ρ_{k+1}), the partial order relation is followed when attacking the system and recording the payoff φ; also, the convergence property of KF is checked (Line 8-11). If these constraints are satisfied, the potential ρ_{k+1} is accepted and added as the child node of ρ_k. Once the tree is constructed, we continue simulating the tracks to the end of time, $k = n$, (Lines 16-17). Finally, all the paths are compared with the original one to select the most deviated path, satisfying the payoff constraint (Lines 18-19).

7.4.3 Heuristic Algorithm based on Sub-optimal Greedy Search

Although the verification algorithm can find the optimal solution, its computation is not polynomial time, and therefore cannot be executed in real-time. A heuristic function can be designed to rank all possible children nodes to select the most likely one. As shown in

Algorithm 6: Verification Algorithm based on BFS

```
Input: LTS model $M$, $n$, $l$, $u$
Output: The most deviated path $\widetilde{\rho}^M$
1  calculate the original path $\rho$ from $k = 0$ to $k = n$
2  set $\rho_l$ as root node
3  for $k$ from $l$ to $u-1$ do
4      for each node $\widetilde{\rho}_k$ in $leaf(\rho_l)$ do
5          find potential observations $Z \leftarrow neighbours(\widetilde{\rho}_k)$
6          for each observed location $\mathbf{z}$ in $Z$ do
7              $\widetilde{\rho}_{k+1} \leftarrow kf(\widetilde{\rho}_k, \mathbf{z})$
8              calculate the attack payoff $\varphi(\widetilde{\rho}_k, \widetilde{\rho}_{k+1})$
9              if $\Gamma(\widetilde{\rho}_k, \widetilde{\rho}_{k+1}) \neq 1$ then
10                 break
11             $\widetilde{\rho}_k = parent(\widetilde{\rho}_{k+1})$
12 $P \leftarrow path(\rho_l)$
13 calculate the path $\widetilde{\rho}$ in set $P$ to $k = n$
14 $\varepsilon = \sum_{k=l+1}^{u} \varphi(\widetilde{\rho}_{k-1}, \widetilde{\rho}_k)/(u-l)$
15 $\widetilde{\rho}^M \leftarrow \arg\max_{\widetilde{\rho} \in P, \varepsilon(\widetilde{\rho}) \leq \theta_{payoff}} dist(\rho, \widetilde{\rho})$
16 return $\widetilde{\rho}^M$
```

Algorithm 7, heuristic search is based on the strategy of making the locally optimal choice at each stage. Compared with the exhaustive exploration, heuristic search can find the sub-optimal solution with significantly less computational cost.

To design a heuristic search algorithm for the optimization problem (7.14), we construct two KFs which run simultaneously for the vehicle tracking. One normal KF accepts the original observations and outputs its states to guide the selection of observations by the adversarial KF. The locally optimal choice is to select the most distant observation of the original one (Line 5). The heuristic function is

$$g(Z, \mathbf{z}) = \arg\max_{\widetilde{\mathbf{z}} \in Z} ||\widetilde{\mathbf{z}} - \mathbf{z}||_2 \tag{7.16}$$

where $\mathbf{z}$ and $\widetilde{\mathbf{z}}$ are the correct and adversarial observation of tracked vehicle obtained in detection, respectively. Lines 7-11 monitor the Bayesian uncertainty and the attack payoff. If the current solution cannot bypass the monitor, the heuristic algorithm will iterate to find the next most distant observation until a feasible one.

7.5 Experimental Results

We conduct a set of experiments to show the effectiveness of our verification algorithm (Algorithm 6) and heuristic search algorithm (Algorithm 7) in the WAMI tracking system.

Algorithm 7: Greedy Based Heuristic Search

```
Input: LTS model $M$, $n$, $l$, $u$
Output: The deviated path $\widetilde{\rho}$
1  calculate the original path $\rho$ from $k = 0$ to $k = n$
2  start from $\widetilde{\rho}_l = \rho_l$
3  for $k$ from $l$ to $u-1$ do
4      find potential observations $Z \leftarrow neighbours(\widetilde{\rho}_k)$
5      $\mathbf{z}(\widetilde{\rho}_k) \leftarrow g(Z, \mathbf{z}(\rho_k))$
6      $\widetilde{\rho}_{k+1} \leftarrow kf(\widetilde{\rho}_k, \mathbf{z}(\widetilde{\rho}_k))$
7      calculate the payoff $\varepsilon \leftarrow \varphi(\widetilde{\rho}_k, \widetilde{\rho}_{k+1})$
8      if $\Gamma(\widetilde{\rho}_k, \widetilde{\rho}_{k+1}) \neq 1$ or $\varepsilon > \theta_{payoff}$ then
9          remove $\mathbf{z}(\widetilde{\rho}_k)$ from $Z$
10         jump to Line 5
11 calculate the path $\widetilde{\rho}$ to $k = n$
12 return $\widetilde{\rho}$
```

We believe our approaches can be generalised to work with other systems using both Bayes filter(s) and neural networks.

7.5.1 Research Questions

Our evaluation experiments are guided by the following three research questions.

RQ1 Does the awareness to the Bayesian uncertainty and the input perturbation in our algorithms improve the quality of the obtained solutions?

RQ2 Can our algorithms prove the robustness of the system?

RQ3 What are the pros and cons of the two algorithms?

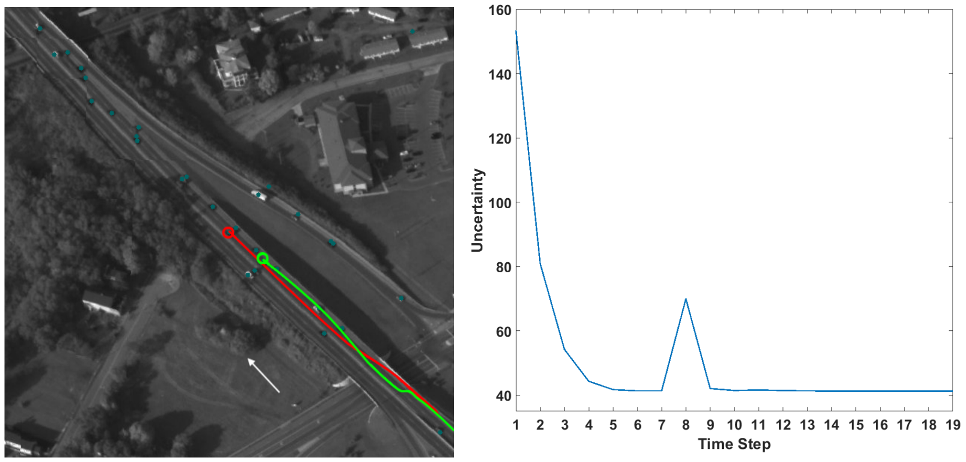
7.5.2 Experimental Setup

We consider a number of original tracks with maximum length of 20 steps ($n = 19, k \in [0, 19]$). An attack on the system is conducted between time steps l and u, denoted as $Attack(l, u)$. The original track is colored in *green* in both the high-resolution images (Figure 7.3–7.4) and the state space unfolding (Figure 7.5). The attacked track is colored in *red*. The white-color arrows in the high-resolution images indicates the ground-truth direction of the vehicle.

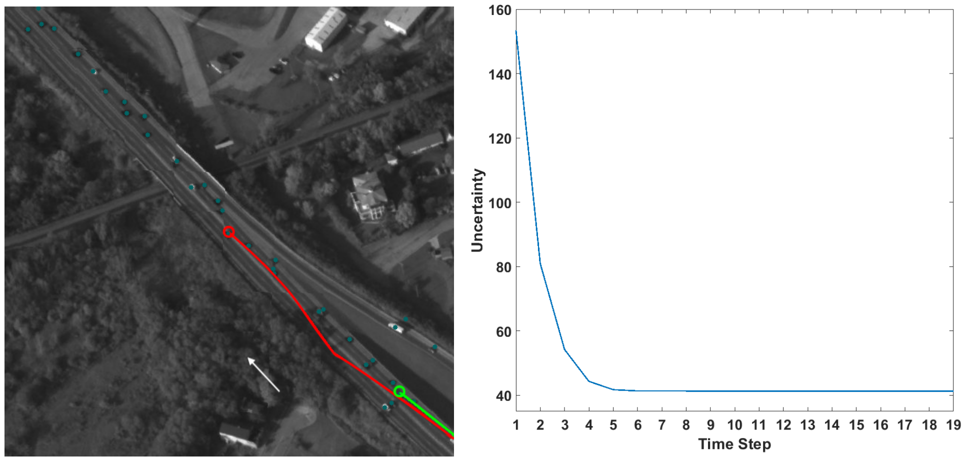
Moreover, in all experiments, we record the following measures for each attack: mean payoff, ε_{avg}, mean deviation, *dist*, as defined in Section 7.3.5, and runtime T (seconds). The payoff threshold is set at $\theta_{payoff} = 1$ and the robustness threshold is set at $\theta_{robustness} = 100$. The thresholds are hyper-parameters and can also be user-defined. Here, we run 100 test scenes and set the mean values for the thresholds.

7.5.3 Returning Good Solutions within Constraints (RQ1)

In Section 7.3.5, we set the the criteria for the adversary to make the attack more realistic. From the adversary's perspective, constraints make sure the attack is not easily detected by simple monitors. In this section, we discuss the impact of these constraints by comparing the two algorithms with the baseline method.



(a) Baseline, $\varepsilon_{avg} = 1.231$, $T = 70$, $dist = 42$



(b) Heuristic/verification, $\varepsilon_{avg} = 0.676$, $T = 73$, $dist = 189$

Figure 7.3: The comparison between the baseline and the heuristic/verification in selected scene with configuration $Attack(5, 8)$

Figure 7.3 presents two plots displaying the change of uncertainty ϵ over time, for baseline method and heuristic/verification algorithm, respectively. In this test scene, heuristic and verification algorithms output the same results. Since the baseline method does not take into consideration the KF's convergence, there exists the situation that, in some time step, no observations are available within the search range – because the only possible observation is attacked – and there is a significant fluctuation at step 8 in the plot.

For the impact of input perturbations, we can see that the heuristic/verification algorithm has a much smaller ε_{avg} – representing a lower mean perturbation cost – than the baseline method. The runtime T is related to the attacking strategy. The heuristic/verification algorithm may need to attack multiple images at each time step to make sure that the remotest observation is taken as the observation. Therefore, the generation of perturbations to attack neural networks may take slightly more time than the baseline method. Overall, the heuristic/verification algorithm can find better attacking solution with smaller distance to the original track than the baseline method, and at the same time it satisfies the constraints from WAMI monitors.

Verification and heuristic search can find the high quality deviated paths, satisfying the constraints of input perturbation and WAMI monitoring.

7.5.4 Proof of Robustness Against the Attack (RQ2)

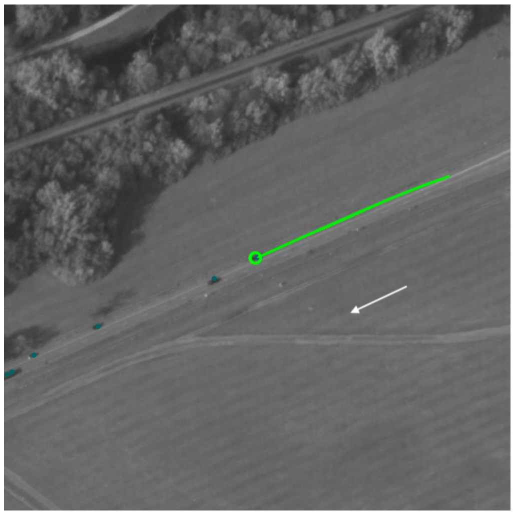
In addition to the ability of finding counterexamples to the robustness property, one may be interested in whether or not our approach can prove, with guarantee, that a system is robust. Figure 7.4 provides an example. We choose the test scene in Figure 7.4b, where the red numbers represent the detected moving vehicles. We apply heuristic search and verification algorithms on the track of vehicle No.0, which is shown in Figure 7.4a. In this case, the WAMI system shows the robustness against the attack. Fundamentally, there are few other vehicles around the tracking car and, at each attack step, only one observation is available in the search range, shown in Figure 7.4c. Thus, to build the tree diagram for this problem, there is only one path to be traversed and therefore no adversarial path is possible.

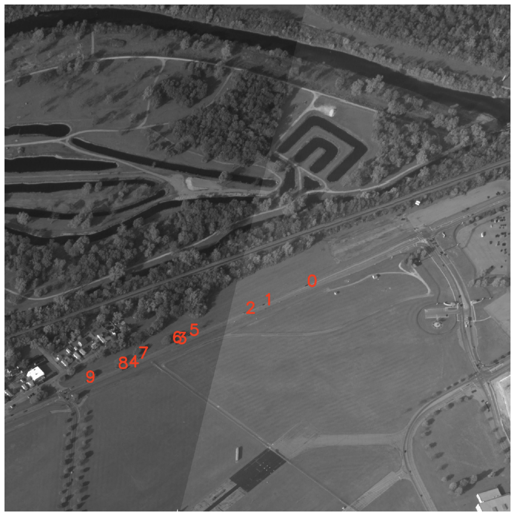
While the above seems to be an extreme case, it actually represents a typical class of systems that tend to be more robust than others, i.e., systems for the test scenes with few external disturbances. The external disturbance comes from the observations of the surrounding vehicles, providing the wrong measurements when KF updates. We remark that this proof can be generalised to exercise the system's robustness for more complicated environments, consisting of an extensive number of vehicles.

WAMI system is robustness against adversarial attack under the circumstance when there are few observations of surrounding vehicles.

7.5.5 Pros and Cons of the Two Algorithms (RQ3)

We compare the performance of the two algorithms – verification and heuristic – by choosing the same tracking start point and apply the algorithms to the same time interval. The detailed running results are presented in Figure 7.5 and Table 7.2. As depicted in Figure 7.5c, the verification method enumerates all possible tracks and selects the most deviated


(a) Output



(b) Detected moving vehicles

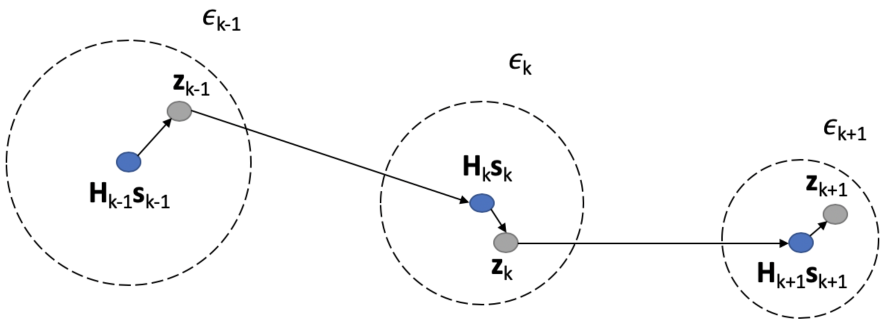
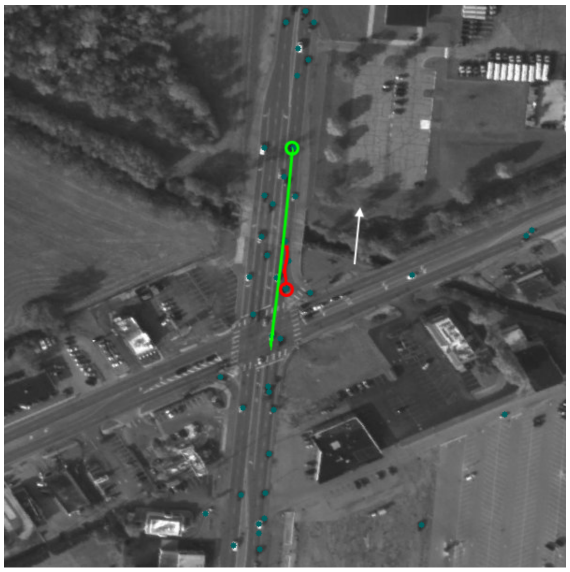

ϵ_{k-1}
$\mathbf{z}_{k-1}$
$\mathbf{H}_{k-1}\mathbf{s}_{k-1}$
ϵ_k
$\mathbf{H}_k\mathbf{s}_k$
$\mathbf{z}_k$
ϵ_{k+1}
$\mathbf{z}_{k+1}$
$\mathbf{H}_{k+1}\mathbf{s}_{k+1}$
(c) Transition of path for track in (a)

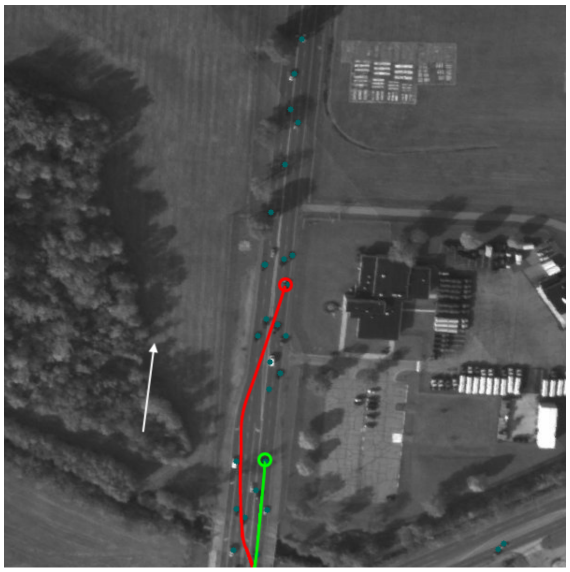
Figure 7.4: Failure in finding an attacked path

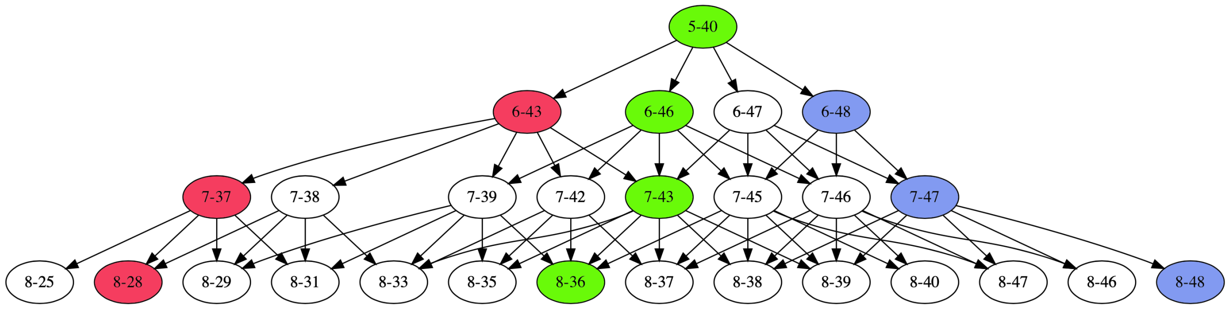
one, colored red. This is the optimal solution to the optimization problem. While the heuristic search method can find a sub-optimal track, colored blue.

Table 7.2: Tracks Generated by Different Algorithms.

Original Track	t	5	6	7	8		19
	ID	40	46	43	36	...	11
	$dist$	0	0	0	0		0
Adv. Track under heuristic search	t	5	6	7	8		19
	ID	40	48	47	48	...	58
	$dist$	0	3.4	10.1	19.7		170.3
Adv. Track under verification	t	5	6	7	8		19
	ID	40	43	37	28	...	7
	$dist$	0	0.4	5.1	13.7		193.3


(a) Heuristic search


(b) Verification



(c) Enumeration of all possible Tracks

Figure 7.5: Heuristic search and verification $Attack(5, 8)$ on a selected scene. Tree graph exhibits all possible tracks, where green is the original track, blue is the attacked track found by heuristic search, and red is the attacked track found by verification. The labels on the nodes represent "(time step)-(ID of associated detection)".

The data in Table 7.2 more precisely indicates the transition policy of the two algorithms. During the time interval $(5, 8)$, the heuristic search guides the tracking to the locally optimal waypoint, which has larger $dist$ values than the verification method. However, for the long term (over 20 time steps), verification is always able to determine the optimal solution.

We also evaluate the two algorithms statistically by sampling 100 test scenes and calculate the average performance. The results can be found in Table 7.3. Since we incorporate the same adversarial attack algorithm, both verification and heuristic search algorithms have similar average perturbation cost ε_{avg}. However, for the runtime cost T, verification is significantly more time-consuming than the heuristic search. Most test scenes have a high concentration of vehicles, i.e., there are large amount of candidate observations. The runtime

Table 7.3: Statistical Comparison between the Verification and Heuristic Search Algorithms

Algorithm	ε_{avg}	T	$dist$	Probability of Finding Best Adv. Track
heuristic search	0.63	78	93	80%
verification	0.65	3465	117	100%

cost of verification is proportional to the candidate observations. Therefore, while verification can find the complete results, it is not suitable for real-time analysis. In contrast, heuristic search, although unable to guarantee the optimal solution, is efficient in runtime. Actually, in our experiments, for 80% of the cases we studied, the heuristic search algorithm can find the optimal solution.

In terms of the safety risks of the WAMI tracking system, we noticed that, the potential risk from the learning components is non-trivial. For example, we often see e.g., a 3-step attack lead to a significantly deviated tracking – a deviation distance of 117 from the original track on average. This illustrates the lack of robustness within the state estimation system we have verified.

Verification is guaranteed to find the optimal solution but computationally intensive, while heuristic search have high probability of detecting best adversarial track.

Chapter 8

Conclusion

This thesis focuses on the verification and validation of learning-enabled autonomous systems (LESs). For this purpose, we propose the debug testing and evaluation for CNNs, RNNs and ensemble trees, and verification for LESs. This includes the debug testing through converge guided testing (Chapter 3), debug testing from distribution-awareness (Chapter 4), backdoor testing of ensemble tree (Chapter 5), reliability evaluation of ML models (Chapter 6) and practical verification of LESs (Chapter 7). We target at robustness and backdoor, which cause the miss-classification/failure of ML models and LESs.

In the proceedings of this chapter, we conclude our main contributions of the thesis. Section 8.1 summarises the work corresponding to each chapter, Section 8.2 describes the main findings and how proposed verification and validations techniques promote more trustworthy ML and LESs. Finally, Section 8.3 discusses the future research avenues based on existing works.

8.1 Thesis Summary

This thesis proposes a range of verification and validation techniques for ML models and LESs, the concise summaries of each chapter are listed below:

- (Chapter 3) Recurrent neural networks (RNNs) have been applied to a broad range of applications, including natural language processing, drug discovery, and video recognition. Their vulnerability to input perturbation is also known. Aligning with a view from software defect detection, this chapter develops a coverage-guided testing approach to systematically exploit the internal behavior of RNNs, with the expectation that such testing can detect defects with high possibility. Technically, the long short-term memory network (LSTM), a major class of RNNs, is thoroughly studied. A family of three test metrics are designed to quantify not only the values but also the temporal relations (including both stepwise and bounded-length) exhibited when LSTM processing inputs. A genetic algorithm is applied to efficiently generate test cases for improving the coverage rate.

- (Chapter 4) We propose a new robustness testing approach for detecting adversarial examples (AEs) that considers both the input distribution and the perceptual quality of inputs. The two considerations are encoded by a novel hierarchical mechanism. First, at the feature level, the input data distribution is extracted and approximated by data compression techniques and probability density estimators. Such quantified feature level distribution, together with indicators that are highly correlated with local robustness, are considered in selecting test seeds. Given a test seed, we then develop a two-step genetic algorithm for local test case generation at the pixel level, in which two fitness functions work alternatively to control the quality of detected AEs.

- (Chapter 5) As the increasing use of machine learning models in security-critical applications, the embedding and extraction of malicious knowledge are equivalent to the notorious backdoor attack and defence, respectively. This chapter studies the embedding and extraction of knowledge in tree ensemble classifiers, and focuses on knowledge expressible with a generic form of Boolean formulas, e.g., backdoor attacks. For the embedding, it is required to be preservative (the original performance of the classifier is preserved), verifiable (the knowledge can be attested), and stealthy (the embedding cannot be easily detected). To facilitate this, we propose two novel, and effective embedding algorithms, one of which is for black-box settings and the other for white-box settings. The embedding can be done in PTIME. Beyond the embedding, we develop the testing algorithm to extract the embedded knowledge from compromised ML models, by reducing the problem to be solvable with an SMT (satisfiability modulo theories) solver. While this novel algorithm can successfully extract knowledge, the reduction leads to an NP computation.

- (Chapter 6) Deep Leaning raises new challenges regarding its reliability in critical functions. In this chapter, we present a model-agnostic reliability assessment method for DL classifiers, based on evidence from robustness evaluation and the operational profile (OP) of a given application. We partition the input space into small cells and then "assemble" their robustness (to the ground truth) according to the OP, where estimators on the cells' robustness and OPs are provided. Reliability estimates in terms of the probability of misclassification per input (pmi) can be derived together with confidence levels.

- (Chapter 7) We study for the first time the verification problem on learning-enabled state estimation systems for robotics, which use Bayes filter for localisation, and use deep neural network to process sensory input into observations for the Bayes filter. Specifically, we are interested in a robustness property of the systems: given a certain ability to an adversary for it to attack the neural network without being noticed, whether or not the state estimation system is able to function with only minor loss of localisation precision? For verification purposes, we reduce the state estimation systems to a novel class of labelled transition systems with payoffs and partial order relations, and formally express the robustness property as a constrained optimisation objective. Based on this, practical verification algorithms are developed.

8.2 Contributions and Main Findings

we re-emphasize the contributions for each chapter and utilize the following research questions to summarize the main findings for each work.

1. Why coverage guided testing is useful in analysing RNNs?

   Instead of identifying a particular type of defects, such as adversarial samples, coverage-guided testing is to generate a set of test cases as diversified as possible while preserving the naturalness, in order to exploit the internal behavior of the neural networks that has real operational impact. The proposed coverage metrics in chapter 3 are of such desirable features of being diverse and natural—with increased coverage, our approach is more likely to find different types of faulty behaviors (e.g., adversarial samples and backdoor samples) that manifest at multiple small regions in the input space, rather than adversarial samples clustered in one region as what normally attack-based methods find.

2. What is the main advantage of hierarchical distribution-aware testing?

   Hierarchical consideration is more effective to detect high-quality (valid) adversarial examples, free of oracle issues, with higher feature distribution probabilities and perception quality, compared with state-of-the-art that either disregards any data distribution or only considers a single (non-hierarchical) distribution. Hierarchical distribution-aware testing contributes more to the growth of DL models' operational robustness, while mitigating the drop of train/test accuracy during adversarial fine-tuning.

3. What is the inherent difficulty for backdoor testing on ensemble tree?

   The knowledge embedding can be done in PTIME. Beyond the embedding, we develop testing algorithm to extract the embedded knowledge, by reducing the problem to be solvable with an SMT (satisfiability modulo theories) solver. While this novel algorithm can successfully extract knowledge, the reduction leads to an NP computation. Therefore, if applying embedding as backdoor attacks and extraction as backdoor testing, results suggest a complexity gap (P vs. NP) between the attack and testing when working with tree ensemble classifiers, which leads to a security concern that a tree ensemble classifier is much easier to be attacked than tested.

4. Why proposed reliability assessment method (RAM) is better than usual accuracy testing?

   An intuitive way of perceiving our RAM, comparing with the usual accuracy testing, is that we enlarge the testing dataset with more test cases around "seeds" (original datapoints in the test set). We determine the oracle of a new test case according to its seed's label and the r-distance. Those enlarged test results form the robustness evidence, and how much they contribute to the overall reliability is proportional to its operational profile. Consequently, exposing to more tests (robustness evaluation) and being more representative of how it will be used (the operational profile), our RAM is more trustworthy. The DL reliability follow the conceptional equation that DL reliability = generalisability × robustness.

5. Can we verify the robustness of learning-enabled state estimation system?

   By formalizing the verification as constrained optimization, and representing partially-ordered label transition system as tree diagram, the verification is reduced to exhaustively search the most deviated path. The rigorous verification is NP-complete since the optimal solution can only be obtained when the tree diagram is unfolded, while the proposed heuristic search algorithm can efficiently find the solution in PTIME. The experiments find for 80% of the cases we studied, the heuristic search algorithm can find the optimal solution as exhaustive search.

To summarize the works in all chapters, we discover that ML models are suffering from a series of risks, which cause the failure of models' prediction. Such failure will propagate to the whole LESs during the interaction between ML models and non-learning components. Although, the failure can be compensated by other non-learning components to some extent, the verification and validation techniques with the objective for evaluation are still necessary for the safe deployment of ML models and LESs. We develop dedicated V&V techniques for different risks of ML models, and find the connection between those techniques in terms of mitigating the same risk. For example, the coverage guided testing can not only detect diverse adversarial samples, but also useful for the backdoor input detection. The backdoor input can be seen as the outlier and effectively detected by OOD detector, which motivate us to further study the similarity and difference between different risks of ML models, and develop the universal V&V techniques for ML models and LESs.

8.3 Future Work

1. Coverage-guided testing has become a controversial research direction. Although, a heap of coverage metrics, such as MC/DC, surprise coverage, are proposed for effectively testing neural networks, some research works argue that structural coverage is not strongly correlated with robustness[189] and neuron-level coverage can be easily satisfied by benign test cases[190]. From my point of view, this should be attributed to the improper design of coverage metrics. The neural network is well-known for its black-box nature, and simple coverage of neuron is meaningless without connection to model's behavior or functionality. Therefore, a promising future work for revitalising coverage guided testing is to develop more interpretable metrics by incorporating the explanation results. In addition, our work in this thesis discovers the preliminary application of coverage guided testing on backdoor detection. The exploration of coverage guided testing on more safety problems, such as privacy, fairness can also be a

significant part of future work. Finally, how to utilize the testing results to mitigate the defects and certify the neural network is also worth to do.

2. For hierarchical distribution-aware testing, the detected on-distribution adversarial examples should be further processed to fix the neural network, so that we can close the loop of "detect-fix-assess" as depicted in [191] and then organise all generated evidence as safety cases[192]. What's more, we currently study the adversarial examples within the norm ball, which make the distribution-aware testing less appealing, as the perturbation within norm ball is meaningless in terms of real-world distribution. The next step is to extend our work to more general distribution of perturbation, such as the light, weather condition and object styles on images. Finally, same as other distribution-aware testing methods, we assume the input data distribution is same as the training data distribution. To relax this assumption, we plan to take distribution-shift into consideration in future versions of hierarchical distribution-aware testing.

3. The extraction of malicious knowledge from compromised tree ensemble is proved as NP-hard. The complexity of knowledge extraction is dominated by the number of joint decision paths, which is exponential to the number of trees. In current work, we reduce the complexity by outlier detection to locate the suspected joint paths, which seems not comprehensive and not realistic. In practice, we may only get comprised tree ensemble without additional data or information. Take this into consideration in the future work, we aims to simplify the tree ensemble model, such as the computation of equivalence classes to derive the feasible joint paths [193], or construction of equivalent single decision tree [194]. The reduction on joint decision paths enables the exact solution by SMT solver, which can provide provable guarantee on knowledge extraction results. Also, the mitigation of extracted knowledge on tree ensemble classifier hasn't been explored yet, the recovery of compromised tree ensemble without damaging model's performance is also challenging.

4. For the reliability evaluation of DL models, We partition the cell in input space and evaluate the reliability of model over the distribution in input space. We not only consider the feature distribution, but also the pixel distribution (the perturbation to seeds input within norm ball). As the community puts more focus on natural disturbance to seeds input, we plan to extend our work with the consideration of robustness to natural disturbance. The natural disturbance is out of the norm ball range and seen as disturbance to features. Therefore, we can partition the cell in latent space, and evaluate the reliability of model over the distribution in latent space. This requires that latent space should encode meaningful features of data and feasible to manipulate with disentangled representation. What's more, we evaluate the DL models before and after adversarial training in experiments and discover the potential of adversarial training to the improvement of reliability. However, as the conceptual expression, reliability equals to generalization times robustness, indicates that new training methods should be developed to better trades off the generalization and robustness for the provable

guarantee on improvement of reliability.

5. We verify the robustness of learning-enabled autonomous systems and find the counterexamples to the robustness property. In [195], we further define the resilience and study the difference and similarity between robustness and resilience of LESs both in theory and experiments. The verification is shown to be NP-complete, which inhibits the practical application of our algorithm on real-world LESs. For the next step, we plan to develop efficient and effective verification methods for both robustness and resilience property of LESs. The runtime verification techniques are worthy of further consideration. A lightweight runtime verification technique will be especially helpful if we intend to work with large-scale, networked systems with hundreds or thousands of components, compared with current off-line analysis. Whats' more, more approaches for the improvement of robustness and resilience are needed. We have investigated two approaches: (1) the utility of joining collaborative components and (2) the utility of a runtime monitor. The exploration of other approaches, and comparison of their relative effectiveness, will be an interesting topic.

Bibliography

[1] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[2] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE international conference on computer vision*, pages 2722–2730, 2015.

[3] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *Proc. of the 26th Symp. on Operating Systems Principles (SOSP)*, pages 1–18, 2017.

[4] Xiaoning Du, Xiaofei Xie, Yi Li, Lei Ma, Yang Liu, and Jianjun Zhao. Deepstellar: Model-based quantitative analysis of stateful deep learning systems. In *Proc. of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 477–487, 2019.

[5] Youcheng Sun, Xiaowei Huang, Daniel Kroening, James Sharp, Matthew Hill, and Rob Ashmore. Testing deep neural networks. *arXiv preprint arXiv:1803.04792*, 2018.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[7] Uday Kamath, John Liu, and James Whitaker. *Deep learning for NLP and speech recognition*, volume 84. Springer, 2019.

[8] Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Mike Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. Achieving human parity in conversational speech recognition. *arXiv preprint arXiv:1610.05256*, 2016.

[9] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning.* MIT press, 2018.

[10] Anders Krogh and John Hertz. A simple weight decay can improve generalization. *Advances in neural information processing systems*, 4, 1991.

[11] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[12] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[13] Daniel A Hashimoto, Guy Rosman, Daniela Rus, and Ozanan R Meireles. Artificial intelligence in surgery: promises and perils. *Annals of surgery*, 268(1):70, 2018.

[14] Qing Rao and Jelena Frtunikj. Deep learning for self-driving cars: Chances and challenges. In *Proceedings of the 1st International Workshop on Software Engineering for AI in Autonomous Systems*, pages 35–38, 2018.

[15] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[16] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *International conference on computer aided verification*, pages 3–29. Springer, 2017.

[17] Mauro Ribeiro, Katarina Grolinger, and Miriam AM Capretz. Mlaas: Machine learning as a service. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pages 896–902. IEEE, 2015.

[18] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1765–1773, 2017.

[19] Stefan Webb, Tom Rainforth, Yee Whye Teh, and M Pawan Kumar. A statistical approach to assessing neural network robustness. In *International Conference on Learning Representations*, 2018.

[20] Fuxun Yu, Zhuwei Qin, Chenchen Liu, Liang Zhao, Yanzhi Wang, and Xiang Chen. Interpreting and evaluating neural network robustness. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 4199–4205, 2019.

[21] Wenjie Ruan, Min Wu, Youcheng Sun, Xiaowei Huang, Daniel Kroening, and Marta Kwiatkowska. Global robustness evaluation of deep neural networks with provable guarantees for the hamming distance. In *International Joint Conferences on Artificial Intelligence Organization*, 2019.

[22] Benjie Wang, Stefan Webb, and Tom Rainforth. Statistically robust neural network classification. In *Uncertainty in Artificial Intelligence*, pages 1735–1745. PMLR, 2021.

[23] Youcheng Sun, Yifan Zhou, Simon Maskell, James Sharp, and Xiaowei Huang. Reliability validation of learning enabled vehicle tracking. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9390–9396. IEEE, 2020.

[24] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[25] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 39–57. IEEE Computer Society, 2017.

[26] Mustafa M Tikir and Jeffrey K Hollingsworth. Efficient instrumentation for code coverage testing. *ACM SIGSOFT Software Engineering Notes*, 27(4):86–96, 2002.

[27] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *proceedings of the 26th Symposium on Operating Systems Principles*, pages 1–18, 2017.

[28] Youcheng Sun, Xiaowei Huang, Daniel Kroening, James Sharp, Matthew Hill, and Rob Ashmore. Deepconcolic: Testing and debugging deep neural networks. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 111–114. IEEE, 2019.

[29] David Berend, Xiaofei Xie, Lei Ma, Lingjun Zhou, Yang Liu, Chi Xu, and Jianjun Zhao. Cats Are Not Fish: Deep Learning Testing Calls for out-of-Distribution Awareness. In *Proc. of the 35th IEEE/ACM Int. Conference on Automated Software Engineering*, ASE'20, pages 1041–1052, New York, NY, USA, 2020. ACM. ISBN 978-1-4503-6768-4. doi: 10.1145/3324884.3416609.

[30] Swaroopa Dola, Matthew B. Dwyer, and Mary Lou Soffa. Distribution-Aware Testing of Neural Networks Using Generative Models. In *IEEE/ACM 43rd Int. Conference on Software Engineering*, ICSE'21, pages 226–237, Madrid, Spain, 2021. IEEE.

[31] Fabrice Harel-Canada, Lingxiao Wang, Muhammad Ali Gulzar, Quanquan Gu, and Miryung Kim. Is Neuron Coverage a Meaningful Measure for Testing Deep Neural Networks? In *Proc. of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 851–862, New York, NY, USA, 2020. ACM.

[32] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

[33] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 707–723. IEEE, 2019.

[34] Paulo Angelo Alves Resende and André Costa Drummond. A survey of random forest based methods for intrusion detection systems. *ACM Computing Surveys (CSUR)*, 51 (3):1–36, 2018.

[35] Maximilian Bachl, Alexander Hartl, Joachim Fabini, and Tanja Zseby. Walling up backdoors in intrusion detection systems. In *Proceedings of the 3rd ACM CoNEXT workshop on big data, machine learning and artificial intelligence for data communication networks*, pages 8–13, 2019.

[36] Robin Bloomfield, Heidy Khlaaf, Philippa Ryan Conmy, and Gareth Fletcher. Disruptive innovations and disruptive assurance: Assuring machine learning and autonomy. *Computer*, 52(9):82–89, 2019.

[37] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Computer Science Review*, 37:100270, 2020.

[38] Bev Littlewood and Lorenzo Strigini. Software reliability and dependability: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 175–188, 2000.

[39] John D. Musa. Operational profiles in software-reliability engineering. *IEEE software*, 10(2):14–32, 1993.

[40] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[41] Nazeeh Ghatasheh. Business analytics using random forest trees for credit risk prediction: a comparison study. *International Journal of Advanced Science and Technology*, 72(2014):19–30, 2014.

[42] Chengwei Liu, Yixiang Chan, Syed Hasnain Alam Kazmi, and Hao Fu. Financial fraud detection model: Based on random forest. *International journal of economics and finance*, 7(7), 2015.

[43] Luckyson Khaidem, Snehanshu Saha, and Sudeepa Roy Dey. Predicting the direction of stock market prices using random forest. *arXiv preprint arXiv:1605.00003*, 2016.

[44] Anne-Laure Boulesteix, Silke Janitza, Jochen Kruppa, and Inke R König. Overview of random forest methodology and practical guidance with emphasis on computational biology and bioinformatics. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(6):493–507, 2012.

[45] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553): 436–444, 2015.

[46] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[47] Ji Young Lee and Franck Dernoncourt. Sequential short-text classification with recurrent and convolutional neural networks. *arXiv preprint arXiv:1603.03827*, 2016.

[48] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. Ieee, 2013.

[49] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, and et al. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Computer Science Review*, 37:100270, 2020. ISSN 1574-0137.

[50] Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. Reachability analysis of deep neural networks with provable guarantees. *arXiv preprint arXiv:1805.02242*, 2018.

[51] Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. Concolic testing for deep neural networks. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 109–119, 2018.

[52] Swaroopa Dola, Matthew B Dwyer, and Mary Lou Soffa. Distribution-aware testing of neural networks using generative models. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 226–237. IEEE, 2021.

[53] Jianmin Guo, Yu Jiang, Yue Zhao, Quan Chen, and Jiaguang Sun. Dlfuzz: Differential fuzzing testing of deep learning systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 739–743, 2018.

[54] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*, pages 303–314, 2018.

[55] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, et al. Deepgauge: Multi-granularity testing criteria for deep learning systems. In *Proce. of the 33rd ACM/IEEE Int. Conference on Automated Software Engineering (ASE'18)*, pages 120–131, 2018.

[56] Chih-Hong Cheng, Chung-Hao Huang, and Hirotoshi Yasuoka. Quantitative projection coverage for testing ml-enabled autonomous systems. In *International Symposium on Automated Technology for Verification and Analysis*, pages 126–142. Springer, 2018.

[57] Jinhan Kim, Robert Feldt, and Shin Yoo. Guiding deep learning system testing using surprise adequacy. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 1039–1049. IEEE, 2019.

[58] Augustus Odena, Catherine Olsson, David Andersen, and Ian Goodfellow. Tensorfuzz: Debugging neural networks with coverage-guided fuzzing. In *International Conference on Machine Learning*, pages 4901–4911. PMLR, 2019.

[59] David Berend. Distribution awareness for ai system testing. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 96–98. IEEE, 2021.

[60] Qiang Hu, Yuejun Guo, Maxime Cordy, Xiaofei Xie, Lei Ma, Mike Papadakis, and Yves Le Traon. An empirical study on data distribution-aware test selection for deep learning enhancement. *ACM Transactions on Software Engineering and Methodology*, 2022.

[61] Taejoon Byun, Abhishek Vijayakumar, Sanjai Rayadurgam, and Darren Cofer. Manifold-based test generation for image classifiers. In *2020 IEEE International Conference On Artificial Intelligence Testing (AITest)*, pages 15–22. IEEE, 2020.

[62] Vincenzo Riccio and Paolo Tonella. Model-based exploration of the frontier of behaviours for deep learning system testing. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 876–888, 2020.

[63] Felipe Toledo, David Shriver, Sebastian Elbaum, and Matthew B Dwyer. Distribution models for falsification and verification of dnns. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 317–329. IEEE, 2021.

[64] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. In *SafeAI@ AAAI*, 2019.

[65] Xijie Huang, Moustafa Alzantot, and Mani Srivastava. Neuroninspect: Detecting backdoors in neural networks via output explanations. *arXiv preprint arXiv:1911.07399*, 2019.

[66] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. Deepinspect: A black-box trojan detection and mitigation framework for deep neural networks. In *IJCAI*, volume 2, page 8, 2019.

[67] Yinpeng Dong, Xiao Yang, Zhijie Deng, Tianyu Pang, Zihao Xiao, Hang Su, and Jun Zhu. Black-box detection of backdoor attacks with limited information and data. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16482–16491, 2021.

[68] Junfeng Guo, Ang Li, and Cong Liu. Aeva: Black-box backdoor detection using adversarial extreme value analysis. *arXiv preprint arXiv:2110.14880*, 2021.

[69] Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao, Cho-Jui Hsieh, and Luca Daniel. Evaluating the robustness of neural networks: An extreme value theory approach. *arXiv preprint arXiv:1801.10578*, 2018.

[70] Zenan Li, Xiaoxing Ma, Chang Xu, Chun Cao, Jingwei Xu, and Jian Lü. Boosting operational dnn testing efficiency through conditioning. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 499–509, 2019.

[71] Antonio Guerriero, Roberto Pietrantuono, and Stefano Russo. Operation is the hardest teacher: estimating dnn accuracy looking for mispredictions. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 348–358. IEEE, 2021.

[72] Tommaso Dreossi, Alexandre Donzé, and Sanjit A Seshia. Compositional falsification of cyber-physical systems with machine learning components. *Journal of Automated Reasoning*, 63(4):1031–1053, 2019.

[73] Cumhur Erkan Tuncali, James Kapinski, Hisahiro Ito, and Jyotirmoy V Deshmukh. Reasoning about safety of learning-enabled components in autonomous cyber-physical systems. *arXiv preprint arXiv:1804.03973*, 2018.

[74] Radoslav Ivanov, James Weimer, Rajeev Alur, George J. Pappas, and Insup Lee. Verisig: Verifying safety properties of hybrid systems with neural network controllers. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '19, page 169–178, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362825. doi: 10.1145/3302504.3311806. URL https://doi.org/10.1145/3302504.3311806.

[75] Shakiba Yaghoubi and Georgios Fainekos. Gray-box adversarial testing for control systems with machine learning components. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '19, page 179–184, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362825. doi: 10.1145/3302504.3311814. URL https://doi.org/10.1145/3302504.3311814.

[76] Ruixin Niu and Lauren Huie. System state estimation in the presence of false information injection. In *Statistical Signal Processing Workshop (SSP)*, pages 385–388. IEEE, 2012.

[77] Qingyu Yang, Liguo Chang, and Wei Yu. On false data injection attacks against kalman filtering in power system dynamic state estimation. *Security and Communication Networks*, 9(9):833–849, 2016.

[78] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *2nd Int. Conf. on Learning Representations*, 2014.

[79] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. BadNets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 7:47230–47244, 2019.

[80] Ali Shafahi, W. Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *Proc. of the 32nd Int. Conf. on Neural Information Processing Systems (NIPS)*, page 6106–6116, 2018.

[81] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symp. on Security and Privacy (SP)*, pages 3–18, 2017.

[82] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. Generating natural language adversarial examples. In *Proc. of the Conf. on Empirical Methods in Natural Language Processing*, pages 2890–2896, 2018.

[83] Jiazhu Dai, Chuanshuai Chen, and Yufeng Li. A backdoor attack against lstm-based text classification systems. *IEEE Access*, 7:138872–138878, 2019.

[84] Zenan Li, Xiaoxing Ma, Chang Xu, and Chun Cao. Structural coverage criteria for neural networks could be misleading. In *2019 IEEE/ACM 41st Int. Conf. on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, pages 89–92, 2019.

[85] Yizhen Dong, Peixin Zhang, Jingyi Wang, Shuang Liu, Jun Sun, Jianye Hao, Xinyu Wang, Li Wang, Jin Song Dong, and Dai Ting. There is limited correlation between coverage and robustness for deep neural networks. *arXiv preprint arXiv:1911.05904*, 2019.

[86] Larry Brader, Howie Hilliker, and Alan Cameron Wills. *Testing for Continuous Delivery with Visual Studio 2012.* Microsoft patterns & practices, 2012.

[87] Peter Bishop and Andrey Povyakalo. Deriving a frequentist conservative confidence bound for probability of failure per demand for systems with different operational and test profiles. *Reliability Engineering & System Safety*, 158:246–253, 2017.

[88] Yao Ming, Shaozu Cao, Ruixiang Zhang, Zhen Li, Yuanzhe Chen, Yangqiu Song, and Huamin Qu. Understanding hidden memories of recurrent neural networks. In *12th IEEE Conf. on Visual Analytics Science and Technology (VAST)*, pages 13–24, 2017.

[89] Phyllis G. Frankl, Richard G. Hamlet, Bev Littlewood, and Lorenzo Strigini. Evaluating testing methods by delivered reliability. *IEEE Tran. on Software Engineering*, 24 (8):586–601, 1998.

[90] Xingyu Zhao, Kizito Salako, Lorenzo Strigini, Valentin Robu, and David Flynn. Assessing safety-critical systems from operational testing: A study on autonomous vehicles. *Information and Software Technology*, 128:106393, 2020.

[91] Richard G. Hamlet and Ross Taylor. Partition testing does not inspire confidence. *IEEE Tran. on Softw. Engineering*, 16(12):1402–1411, 1990.

[92] Edmund M Clarke Jr, Orna Grumberg, Daniel Kroening, Doron Peled, and Helmut Veith. *Model checking.* The MIT Press, 2018.

[93] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Pranav Patel. Finding motifs in time series. In *Proc. of the 2nd Workshop on Temporal Data Mining*, pages 53–68, 2002.

[94] Francesco Crecchi, Davide Bacciu, and Battista Biggio. Detecting black-box adversarial examples through nonlinear dimensionality reduction. In *27th European Symp. on Artificial Neural Networks*, 2019.

[95] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. In *Workshop on Artificial Intelligence Safety 2019 co-located with the 33rd AAAI Conf. on Artificial Intelligence*, volume 2301, 2019.

[96] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proc. of the IEEE Conf. on computer vision and pattern recognition*, pages 2818–2826, 2016.

[97] Zhiqiang Gong, Ping Zhong, and Weidong Hu. Diversity in machine learning. *IEEE Access*, 7:64323–64350, 2019.

[98] Yang Yu, Yu-Feng Li, and Zhi-Hua Zhou. Diversity regularized machine. In *22nd Int. Joint Conf. on Artif. Intel. (IJCAI)*, pages 1603–1608, 2011.

[99] Nicolas Papernot, Patrick McDaniel, Ananthram Swami, and Richard Harang. Crafting adversarial input sequences for recurrent neural networks. In *IEEE Military Communications Conf. (MILCOM)*, pages 49–54. IEEE, 2016.

[100] Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chem. Sci.*, 9:513–530, 2018.

[101] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human action classes from videos in the wild. *CRCV-TR-12-01*, 2012.

[102] Jason Wei and Kai Zou. EDA: Easy data augmentation techniques for boosting performance on text classification tasks. In *Proc. of the 2019 Conf. on Empirical Methods in Natural Language Processing and the 9th Int. Joint Conf. on Natural Language Processing (EMNLP-IJCNLP)*, pages 6382–6388. Association for Computational Linguistics, 2019.

[103] RDKit, online. RDKit: Open-source cheminformatics. `http://www.rdkit.org`, 2013. [Online; accessed 11-April-2013].

[104] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Towards practical verification of machine learning: The case of computer vision systems. *arXiv preprint arXiv:1712.01785*, 2017.

[105] Melika Behjati, Seyed-Mohsen Moosavi-Dezfooli, Mahdieh Soleymani Baghshah, and Pascal Frossard. Universal adversarial attacks on text classifiers. In *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, pages 7345–7349. IEEE, 2019.

[106] David Lane, David Bisset, Rob Buckingham, Geoff Pegman, and Tony Prescott. New foresight review on robotics and autonomous systems. Technical Report No. 2016.1, LRF, 2016.

[107] J. M. Zhang, M. Harman, L. Ma, and Y. Liu. Machine Learning Testing: Survey, Landscapes and Horizons. *IEEE Tran. on Software Engineering*, 2020. doi: 10.1109/TSE.2019.2962027. Early access.

[108] David Berend. Distribution awareness for AI system testing. In *43rd IEEE/ACM Int. Conf. on Software Engineering: Companion Proceedings, ICSE Companion 2021, Madrid, Spain, May 25-28, 2021*, pages 96–98. IEEE, 2021.

[109] Felipe Toledo, David Shriver, Sebastian Elbaum, and Matthew B Dwyer. Distribution models for falsification and verification of dnns. In *IEEE/ACM Int. Conf. on Automated Software Engineering (ASE'21)*, 2021.

[110] Taejoon Byun, Abhishek Vijayakumar, Sanjai Rayadurgam, and Darren Cofer. Manifold-based Test Generation for Image Classifiers. In *Int. Conf. On Artificial Intelligence Testing (AITest)*, pages 15–22, Oxford, UK, 2020. IEEE. doi: 10.1109/AITEST49225.2020.00010.

[111] Yue Zhong, Lizhuang Liu, Dan Zhao, and Hongyang Li. A generative adversarial network for image denoising. *Multimedia Tools and Applications*, 79(23):16517–16529, 2020.

[112] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *Computer Aided Verification*, volume 10426 of *LNCS*, pages 3–29, Cham, 2017. Springer International Publishing.

[113] Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. Reachability Analysis of Deep Neural Networks with Provable Guarantees. In *Proceedings of the Twenty-Seventh Int. Joint Conference on Artificial Intelligence, IJCAI-18*, pages 2651–2659. Int. Joint Conferences on Artificial Intelligence Organization, 2018.

[114] Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao, Cho-Jui Hsieh, and Luca Daniel. Evaluating the robustness of neural networks: An extreme value theory approach. In *6th Int. Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[115] Stefan Webb, Tom Rainforth, Yee Whye Teh, and M. Pawan Kumar. A statistical approach to assessing neural network robustness. In *ICLR'19*, New Orleans, LA, USA, 2019.

[116] Moustafa Alzantot, Yash Sharma, Supriyo Chakraborty, Huan Zhang, Cho-Jui Hsieh, and Mani B Srivastava. Genattack: Practical black-box attacks with gradient-free optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1111–1119, 2019.

[117] Chenwang Wu, Wenjian Luo, Nan Zhou, Peilan Xu, and Tao Zhu. Genetic algorithm with multiple fitness functions for generating adversarial examples. In *IEEE Congress on Evolutionary Computation, CEC 2021, Kraków, Poland, June 28 - July 1, 2021*, pages 1792–1799. IEEE, 2021.

[118] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, pages 1–18. ACM, 2017.

[119] Youcheng Sun, Xiaowei Huang, Daniel Kroening, James Sharp, Matthew Hill, and Rob Ashmore. Deepconcolic: testing and debugging deep neural networks. In *Proceedings of the 41st Int. Conference on Software Engineering: Companion Proceedings, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, pages 111–114. IEEE / ACM, 2019.

[120] Wei Huang, Youcheng Sun, Xingyu Zhao, James Sharp, Wenjie Ruan, Jie Meng, and Xiaowei Huang. Coverage guided testing for recurrent neural networks. *IEEE Tran. on Reliability*, 2021.

[121] Shenao Yan, Guanhong Tao, Xuwei Liu, Juan Zhai, Shiqing Ma, Lei Xu, and Xiangyu Zhang. Correlations between Deep Neural Network Model Coverage Criteria and Model Quality. In *Proc. of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, pages 775–787, New York, NY, USA, 2020. ACM.

[122] Lily Weng, Pin-Yu Chen, Lam Nguyen, Mark Squillante, Akhilan Boopathy, Ivan Oseledets, and Luca Daniel. PROVEN: Verifying robustness of neural networks with a probabilistic approach. In *ICML'19*, volume 97, pages 6727–6736. PMLR, 2019.

[123] Yao-Yuan Yang, Cyrus Rashtchian, Hongyang Zhang, Russ R Salakhutdinov, and Kamalika Chaudhuri. A Closer Look at Accuracy vs. Robustness. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33 of *NeurIPS'20*, pages 8588–8601. Curran Associates, Inc., 2020.

[124] Yisen Wang, Xingjun Ma, James Bailey, Jinfeng Yi, Bowen Zhou, and Quanquan Gu. On the convergence and robustness of adversarial training. In *Proceedings of the 36th Int. Conference on Machine Learning, ICML'19*, volume 97, pages 6586–6595, Long Beach, California, USA, 2019. PMLR.

[125] Jingyi Wang, Jialuo Chen, Youcheng Sun, Xingjun Ma, Dongxia Wang, Jun Sun, and Peng Cheng. Robot: robustness-oriented testing for deep learning systems. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 300–311. IEEE, 2021.

[126] Jingyi Wang, Jialuo Chen, Youcheng Sun, Xingjun Ma, Dongxia Wang, Jun Sun, and Peng Cheng. RobOT: Robustness-Oriented Testing for Deep Learning Systems. In *2021 IEEE/ACM 43rd Int. Conf. on Software Engineering (ICSE'21)*, pages 300–311, 2021. doi: 10.1109/ICSE43902.2021.00038.

[127] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric Xing, Laurent El Ghaoui, and Michael Jordan. Theoretically principled trade-off between robustness and accuracy. In *International conference on machine learning*, pages 7472–7482. PMLR, 2019.

[128] Taejoon Byun and Sanjai Rayadurgam. Manifold-based test generation for image classifiers. In *ICSE '20: 42nd Int. Conference on Software Engineering, Workshops, Seoul, Republic of Korea, 27 June - 19 July, 2020*, page 221. ACM, 2020. doi: 10.1145/3387940.3391460. URL `https://doi.org/10.1145/3387940.3391460`.

[129] Vincenzo Riccio and Paolo Tonella. Model-based exploration of the frontier of behaviours for deep learning system testing. In Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann, editors, *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, pages 876–888. ACM, 2020. doi: 10.1145/3368089.3409730. URL `https://doi.org/10.1145/3368089.3409730`.

[130] Hossein Hosseini and Radha Poovendran. Semantic adversarial examples. In *2018 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 1614–1619. IEEE Computer Society, 2018.

[131] Zhengli Zhao, Dheeru Dua, and Sameer Singh. Generating natural adversarial examples. In *6th Int. Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[132] Alain Hore and Djemel Ziou. Image quality metrics: Psnr vs. ssim. In *2010 20th international conference on pattern recognition*, pages 2366–2369. IEEE, 2010.

[133] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Trans. Image Process.*, 13(4):600–612, 2004.

[134] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6626–6637, 2017.

[135] S. H. Lokerse, L. P. J. Veelenturf, and J. G. Beltman. Density estimation using SOFM and adaptive kernels. In *Neural Networks: Artificial Intelligence and Industrial Applications - Proceedings of the Third Annual SNN Symposium on Neural Networks, Nijmegen, The Netherlands, September 14-15, 1995*, pages 203–206. Springer, 1995.

[136] Xingyu Zhao, Wei Huang, Alec Banks, Victoria Cox, David Flynn, Sven Schewe, and Xiaowei Huang. Assessing the Reliability of Deep Learning Classifiers Through Robustness Evaluation and Operational Profiles. In *AISafety'21 Workshop at IJCAI'21*, volume 2916, 2021.

[137] Ahmadreza Jeddi, Mohammad Javad Shafiee, and Alexander Wong. A simple fine-tuning is all you need: Towards robust deep learning via adversarial fine-tuning. In *Workshop on Adversarial Machine Learning in Real-World Computer Vision Systems and Online Challenges (AML-CV) @ CVPR'21*, pages 1–5, 2021.

[138] Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *EMNLP-CoNLL 2007, Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, June 28-30, 2007, Prague, Czech Republic*, pages 410–420. ACL, 2007.

[139] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *Proc. of the 37th Int. Conf. on Machine Learning (ICML'20)*, volume 119, pages 2206–2216. PMLR, 2020.

[140] David W Scott. Feasibility of multivariate density estimates. *Biometrika*, 78(1):197–205, 1991.

[141] Han Liu, John Lafferty, and Larry Wasserman. Sparse nonparametric density estimation in high dimensions using the rodeo. In *Artificial Intelligence and Statistics*, pages 283–290. PMLR, 2007.

[142] Maximilian Bachl, Alexander Hartl, Joachim Fabini, and Tanja Zseby. Walling up backdoors in intrusion detection systems. *Proceedings of the 3rd ACM CoNEXT Workshop on Big Data, Machine Learning and Artificial Intelligence for Data Communication Networks*, pages 8–13, 2019.

[143] Y. Chen, X. Gong, Q. Wang, X. Di, and H. Huang. Backdoor attacks and defenses for deep neural networks in outsourced cloud environments. *IEEE Network*, 34(5): 141–147, 2020. doi: 10.1109/MNET.011.1900577.

[144] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 7:47230–47244, 2019.

[145] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *CoRR*, abs/1712.05526, 2017. URL `http://arxiv.org/abs/1712.05526`.

[146] Min Du, Ruoxi Jia, and Dawn Song. Robust Anomaly Detection and Backdoor Attack Detection Via Differential Privacy. In *International Conference on Learning Representations (ICLR)*, 2020.

[147] Alex Kantchelian, J. D. Tygar, and Anthony D. Joseph. Evasion and hardening of tree ensemble classifiers. In *Proceedings of the 33nd International Conference on Machine Learning*, volume 48, pages 2387–2396, 2016.

[148] GG Moisen. Classification and regression trees. *In: Jørgensen, Sven Erik; Fath, Brian D.(Editor-in-Chief). Encyclopedia of Ecology, volume 1. Oxford, UK: Elsevier. p. 582-588.*, pages 582–588, 2008.

[149] Arthur Asuncion and David Newman. Uci machine learning repository, 2007.

[150] Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):1–27, 2011.

[151] Jerry L Hintze and Ray D Nelson. Violin plots: a box plot-density trace synergism. *The American Statistician*, 52(2):181–184, 1998.

[152] Floriana Esposito, Donato Malerba, Giovanni Semeraro, and Valentina Tamma. The effects of pruning methods on the predictive accuracy of induced decision trees. *Applied Stochastic Models in Business and Industry*, 15(4):277–299, 1999.

[153] Ximing Qiao, Yukun Yang, and Hai Li. Defending neural backdoors via generative distribution modeling. In *Advances in Neural Information Processing Systems*, pages 14004–14013, 2019.

[154] Robin Bloomfield, Heidy Khlaaf, Philippa Ryan Conmy, and Gareth Fletcher. Disruptive innovations and disruptive assurance: Assuring machine learning and autonomy. *Computer*, 52(9):82–89, 2019. ISSN 0018-9162.

[155] Bev Littlewood and Lorenzo Strigini. Software reliability and dependability: A roadmap. In *Proc. of the Conf. on The Future of Software Engineering*, ICSE 2000, pages 175–188, 2000.

[156] John Musa. Operational profiles in software-reliability engineering. *IEEE Software*, 10 (2):14–32, 1993. ISSN 0740-7459.

[157] Xingyu Zhao, Alec Banks, James Sharp, Valentin Robu, David Flynn, Michael Fisher, and Xiaowei Huang. A Safety Framework for Critical Systems Utilising Deep Neural Networks. In António Casimiro, Frank Ortmeier, Friedemann Bitsch, and Pedro Ferreira, editors, *Computer Safety, Reliability, and Security*, volume 12234 of *LNCS*, pages 244–259, Cham, 2020. Springer Int. Publishing.

[158] Lorenzo Strigini and Bev Littlewood. Guidelines for statistical testing. Technical report, City, University of London, 1997. URL `http://openaccess.city.ac.uk/254/`.

[159] Nicholas Carlini and David Wagner. Towards Evaluating the Robustness of Neural Networks. In *IEEE Symp. on Security and Privacy (SP)*, pages 39–57, San Jose, CA, USA, 2017. IEEE. doi: 10.1109/SP.2017.49.

[160] Peter Bishop, Robin Bloomfield, Bev Littlewood, Andrey Povyakalo, and David Wright. Toward a formalism for conservative claims about the dependability of software-based systems. *IEEE Transactions on Software Engineering*, 37(5):708–717, 2011.

[161] Lorenzo Strigini and Andrey Povyakalo. Software fault-freeness and reliability predictions. In *SafeComp'13*, volume 8153 of *LNCS*, pages 106–117, Berlin, Heidelberg, 2013. Springer. ISBN 978-3-642-40793-2.

[162] Gero Walter and Thomas Augustin. Imprecision and prior-data conflict in generalized Bayesian inference. *Journal of Statistical Theory & Practice*, 3(1):255–271, 2009.

[163] Xingyu Zhao, Valentin Robu, David Flynn, Fateme Dinmohammadi, Michael Fisher, and Matt Webster. Probabilistic model checking of robots deployed in extreme environments. In *AAAI'19*, volume 33, pages 8076–8084, 2019.

[164] Roberto Pietrantuono, Peter Popov, and Stefano Russo. Reliability assessment of service-based software under operational profile uncertainty. *Reliability Engineering & System Safety*, 204:107193, 2020.

[165] Bernard W Silverman. *Density estimation for statistics and data analysis*, volume 26. CRC press, 1986.

[166] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. of Machine Learning Research*, 13(2):281–305, 2012.

[167] Yen-Chi Chen. A tutorial on kernel density estimation and recent advances. *Biostatistics & Epidemiology*, 1(1):161–187, 2017.

[168] Antonio Guerriero. Reliability Evaluation of ML systems, the oracle problem. In *ISSREW'20*, pages 127–130, Coimbra, Portugal, 2020. IEEE. doi: 10.1109/ISSREW51248.2020.00050.

[169] David W Scott. *Multivariate density estimation: theory, practice, and visualization*. John Wiley & Sons, 2015.

[170] Arturs Backurs, Piotr Indyk, and Tal Wagner. Space and time efficient kernel density estimation in high dimensions. In *NeurIPS'19*, pages 15773–15782, 2019.

[171] Philip R Bevington, D Keith Robinson, J Morris Blair, A John Mallinckrodt, and Susan McKay. *Data reduction and error analysis for the physical sciences*, volume 7. American Institute of Physics, 1993.

[172] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

[173] Erfan Asaadi, Ewen Denney, Jonathan Menzies, Ganesh J. Pai, and Dimo Petroff. Dynamic Assurance Cases: A Pathway to Trusted Autonomy. *Computer*, 53(12):35–46, 2020. doi: 10.1109/MC.2020.3022030.

[174] Abdulhakim Qahtan, Suojin Wang, and Xiangliang Zhang. KDE-Track: An Efficient Dynamic Density Estimator for Data Streams. *IEEE Transactions on Knowledge and Data Engineering*, 29(3):642–655, 2017. doi: 10.1109/TKDE.2016.2626441.

[175] Xingyu Zhao, Radu Calinescu, Simos Gerasimou, Valentin Robu, and David Flynn. Interval change-point detection for runtime probabilistic model checking. In *ASE'20*, pages 163–174. IEEE/ACM, 2020.

[176] Ziyuan Zhong, Yuchi Tian, and Baishakhi Ray. Understanding Local Robustness of Deep Neural Networks under Natural Variations. In *FASE'21*, pages 313–337, 2021.

[177] Keith W. Miller, Larry J. Morell, Robert E. Noonan, Stephen K. Park, David M. Nicol, Branson W. Murrill, and M Voas. Estimating the probability of failure when testing reveals no failures. *IEEE Transactions on Software Engineering*, 18(1):33–43, 1992. ISSN 0098-5589.

[178] Georgios Papadopoulos, Maurice F Fallon, John J Leonard, and Nicholas M Patrikalakis. Cooperative localization of marine vehicles using nonlinear state estimation. In *IROS*, pages 4874–4879. IEEE, 2010.

[179] Neil Gordon, David Salmond, and Craig Ewing. Bayesian state estimation for tracking and guidance using the bootstrap filter. *Journal of Guidance, Control, and Dynamics*, 18(6):1434–1443, 1995.

[180] Y. Zhou and S. Maskell. Detecting and tracking small moving objects in wide area motion imagery (wami) using convolutional neural networks (cnns). In *2019 22th International Conference on Information Fusion (FUSION)*, pages 1–8, July 2019.

[181] Wei He and Yiting Dong. Adaptive fuzzy neural network control for a constrained robot using impedance learning. *IEEE transactions on neural networks and learning systems*, 2017.

[182] Youcheng Sun, Yifan Zhou, Simon Maskell, James Sharp, and Xiaowei Huang. Reliability validation of learning enabled vehicle tracking. In *ICRA*, 2020.

[183] R. E. Kalman. A new approach to linear filtering and prediction problems. *J. Basic Eng.*, 82(1):35–45, 1960. doi: 10.1109/ijcnn.2011.6033589.

[184] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[185] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *CAV'17*, volume 10426 of *LNCS*, pages 3–29. Springer, 2017. ISBN 978-3-319-63387-9.

[186] Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. Reachability analysis of deep neural networks with provable guarantees. In *IJCAI2018*, pages 2651–2659, 2018.

[187] Wenjie Ruan, Min Wu, Youcheng Sun, Xiaowei Huang, Daniel Kroening, and Marta Kwiatkowska. Global robustness evaluation of deep neural networks with provable guarantees for the hamming distance. In *IJCAI2019*, pages 5944–5952, 2019.

[188] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2574–2582, 2016.

[189] Yizhen Dong, Peixin Zhang, Jingyi Wang, Shuang Liu, Jun Sun, Jianye Hao, Xinyu Wang, Li Wang, Jin Song Dong, and Dai Ting. There is limited correlation between coverage and robustness for deep neural networks. *arXiv preprint arXiv:1911.05904*, 2019.

[190] Zenan Li, Xiaoxing Ma, Chang Xu, and Chun Cao. Structural coverage criteria for neural networks could be misleading. In *2019 IEEE/ACM 41st Int. Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, pages 89–92. IEEE, 2019.

[191] Xingyu Zhao, Wei Huang, Sven Schewe, Yi Dong, and Xiaowei Huang. Detecting operational adversarial examples for reliable deep learning. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*, pages 5–6. IEEE, 2021.

[192] Xingyu Zhao, Alec Banks, James Sharp, Valentin Robu, David Flynn, Michael Fisher, and Xiaowei Huang. A safety framework for critical systems utilising deep neural networks. In *International Conference on Computer Safety, Reliability, and Security*, pages 244–259. Springer, 2020.

[193] John Törnblom and Simin Nadjm-Tehrani. Formal verification of input-output mappings of tree ensembles. *Science of Computer Programming*, 194:102450, 2020.

[194] Thibaut Vidal and Maximilian Schiffer. Born-again tree ensembles. In *International conference on machine learning*, pages 9743–9753. PMLR, 2020.

[195] Wei Huang, Yifan Zhou, Youcheng Sun, Alec Banks, Jie Meng, James Sharp, Simon Maskell, and Xiaowei Huang. Formal verification of robustness and resilience of learning-enabled state estimation systems for robotics. *arXiv preprint arXiv:2010.08311*, 2020.