Formal And Geometric Foundations For Deep Learning Under Safety Constraints

by

Marco Casadio

Submitted for the degree of

Doctor of Philosophy

HERIOT-WATT UNIVERSITY

SCHOOL OF MATHEMATICAL AND COMPUTER SCIENCES

September 2025

Abstract

Deep neural networks (DNNs) achieve high accuracy yet can be unreliable under distribution shifts and perturbations. This thesis develops *formal and geometric foundations* for deep learning (DL) under safety constraints: we state desired behaviour of DNNs as properties over regions of the DNN input space and verify them in computer vision, natural language processing (NLP), and network intrusion detection systems (NIDS). We study (i) *local robustness*, label invariance on ε-balls (i.e., the set of inputs in the DNN's input space within distance ε from a reference input), and (ii) *global properties*, classification invariance on hyper-rectangles (i.e., axis-aligned boxes) that encode rules. Our central thesis is that *verifiability*, the ability to formally *define* and *check* a property, hinges on how the property is specified in terms of geometric subspaces: properties become tractable when subspaces are precisely definable and checkable (e.g., ε-balls or hyper-rectangles), and elusive otherwise. Concretely, we: (i) relate common robustness notions in computer vision (CR, SCR, SR, LR) and show that optimising losses that do not match the verified property yields weaker certificates; (ii) in NLP, bound sets of semantically equivalent rephrasings by hyper-rectangles in the embedding space to enable verification despite the embedding gap; and (iii) in NIDS, formalise rule-based global properties and use them for verification and property-driven training. Across domains, aligning training objectives with the verified property improves certified behaviour and out-of-distribution generalisation. This thesis offers a practical recipe: specify the behaviour precisely, build subspaces amenable to formal verification, train for that property, and verify with precise tools, showing that principled specifications provide a practical path to more reliable DL.

To my family and friends, for their unwavering support.

Acknowledgements

I am grateful to my supervisor, Prof. Ekaterina Komendantskaya, for her guidance, patience, and encouragement throughout this work. Her insight shaped the ideas and direction of this thesis. I also thank my colleagues and collaborators for helpful discussions and feedback along the way.

My thanks go to my family and friends for their steady support and encouragement throughout my studies.

Table of Contents

1 Introduction **1**
- 1.1 Motivation and Scope . . . 3
- 1.2 Methodological Framework . . . 5
- 1.3 Thesis Contributions . . . 7
- 1.4 Limitations . . . 9
- 1.5 Contribution Declaration . . . 9
- 1.6 List of Additional Papers . . . 10

2 Background & Literature Review **12**
- 2.1 Background . . . 13
  - 2.1.1 Deep Neural Networks and Data Representations . . . 13
  - 2.1.2 Examples of DNN Applications . . . 14
  - 2.1.3 Adversarial Examples . . . 16
  - 2.1.4 Formal Definitions of Robustness and Training Techniques . . . 18
  - 2.1.5 Geometric Analysis of Embedding Spaces . . . 21
  - 2.1.6 Specification Languages for Neural Network Verification . . . 23
  - 2.1.7 Local vs Global Properties . . . 26
- 2.2 Literature Review . . . 27
  - 2.2.1 Robust Training . . . 27
  - 2.2.2 DNN Verification . . . 29
  - 2.2.3 Robustness and Verification in Computer Vision . . . 32
  - 2.2.4 NLP Robustness . . . 33
  - 2.2.5 Previous NLP Verification Approaches . . . 37
  - 2.2.6 Datasets Used in NLP Verification . . . 39
  - 2.2.7 NIDS Robustness and Verification . . . 41
  - 2.2.8 A Systematic View of Training-Verification Pipelines . . . 43

3 Neural Network Robustness as a Verification Property in Computer Vision **46**
- 3.1 Robustness Evaluation in Practice . . . 47
- 3.2 Relative Comparison of Definitions of Robustness . . . 50
  - 3.2.1 Lipschitz vs Standard Robustness . . . 51
  - 3.2.2 (Strong) Classification Robustness . . . 51
  - 3.2.3 Standard vs Classification Robustness . . . 53
  - 3.2.4 Illustrative Specification in Vehicle . . . 54
- 3.3 Other Properties of Robustness Definitions . . . 55
- 3.4 Conclusions . . . 57

4 Neural Network Robustness as a Verification Property in NLP **59**
- 4.1 The Problem of NLP Verification . . . 61
- 4.2 The Parametric NLP Verification Pipeline . . . 62
  - 4.2.1 Datasets . . . 63
  - 4.2.2 Semantic Perturbations . . . 64
  - 4.2.3 NLP Embeddings . . . 66
  - 4.2.4 Working with Embedding Spaces: Our Approach . . . 67
  - 4.2.5 Training . . . 72

4.2.6 Choice of Verification Algorithm . . . 73
4.3 Characterisation of Verifiable Subspaces . . . 74
4.3.1 Metrics for Understanding the Properties of Embedding Spaces . 75
4.3.2 Baseline Experiments for Understanding the Properties of Embedding Spaces . . . 76
4.3.3 Verifiability-Generalisability Trade-off for Geometric Subspaces . 77
4.3.4 Verifiability-Generalisability Trade-off for Semantic Subspaces . 80
4.3.5 Adversarial Training on Semantic Subspaces . . . 83
4.4 NLP Case Studies . . . 87
4.4.1 Role of False Positives and False Negatives . . . 89
4.4.2 Performance of Existing LLMs as Safety-Critical Filters . . . 90
4.4.3 Experimental Setup of the Verification Pipeline . . . 91
4.4.4 Analysis of the Role of Embedding Functions . . . 93
4.4.5 Analysis of Perturbations . . . 95
4.4.6 Embedding Error . . . 103
4.5 Conclusions and Future Work . . . 106

5 Generalising the Training-Verification Pipeline to NIDS 112
5.1 Methodology . . . 113
5.1.1 Feature Extraction and Input Representation . . . 114
5.1.2 Feature Categorisation . . . 115
5.1.3 Adapted Training-Verification Pipeline . . . 116
5.2 Specifications . . . 118
5.2.1 Global Specification Properties . . . 119
5.2.2 Training on Global Properties . . . 125
5.3 Experiments . . . 127
5.4 Conclusions . . . 132

6 Conclusions 133
6.1 Summary of Contributions . . . 133
6.2 Cross-Domain Insights . . . 134
6.3 Limitations . . . 135
6.4 Guidelines for Practitioners . . . 136
6.5 Future Work . . . 136
6.6 Concluding Remarks . . . 137

References 138

List of Tables

2.1 *Construction complexity for different geometric shapes, where m is the number of dimensions and p is the number of points or generators.* 22
2.2 *Summary of the main features of the existing NLP verification approaches. In bold are state-of-the-art methods.* 35
2.3 *Summary of the main features of the existing randomised smoothing approaches.* 36
2.4 *Summary of the main features of the datasets used in NLP verification.* 40

3.1 *Constraint satisfaction results for the CR, SR, and LR constraints. Calculated over the test set and expressed as percentages.* 52
3.2 *A comparison of the different types of robustness studied in this work. Top half: general properties. Bottom half: relation to existing machine-learning literature* 56

4.1 *Character-level perturbations: their types and examples of how each type acts on a given sentence from the R-U-A-Robot dataset [19]. Perturbations are selected from random words that have 3 or more characters, first and last characters of a word are never perturbed.* 65
4.2 *Word-level perturbations: their types and examples of how each type acts on a given sentence from the R-U-A-Robot dataset [19].* 66
4.3 *Mean and standard deviation of the accuracy of the baseline DNN on the RUAR and the Medical datasets. All experiments are replicated five times.* 77
4.4 *Sets of geometric subspaces used in the experiments, their cardinality and average volumes of hyper-rectangles. All shapes are eigenspace rotated for better precision.* 78
4.5 *Verifiability of the baseline DNN on the RUAR and the Medical datasets, for a selection of geometric subspaces; using the ERAN verifier.* 78
4.6 *Generalisability of the selected geometric subspaces* $\mathbb{H}_{\varepsilon=0.005}$, $\mathbb{H}_{\varepsilon=0.05}$ *and* $\mathbb{H}_{sh}$, *measured on the sets of semantic perturbations* $\mathscr{A}^b_{t_{RAUR}}(\mathscr{D}^{pos})$ *and* $\mathscr{A}^b_{t_{medical}}(\mathscr{D}^{pos})$. 79
4.7 *Sets of semantic subspaces used in the experiments, their cardinality and average volumes of hyper-rectangles. All shapes are eigenspace rotated for better precision.* 81
4.8 *Verifiability of the baseline DNN on the RUAR and the Medical datasets, for a selection of semantic subspaces; using the ERAN verifier.* 81
4.9 *Verifiability of the baseline DNN on the RUAR and the Medical datasets, for a selection of semantic subspaces; using the Marabou verifier.* 81
4.10 *Generalisability of the selected geometric subspaces* $\mathbb{H}_{\varepsilon=0.005}$ *and* $\mathbb{H}_{\varepsilon=0.05}$ *and the semantic subspaces* $\mathbb{H}_{word}$ *and* $\mathbb{H}_{pj}$, *measured on the sets of semantic perturbations* $\mathscr{A}^b_{t_{RUAR}}(\mathscr{D}^{pos})$ *and* $\mathscr{A}^b_{t_{medical}}(\mathscr{D}^{pos})$. *Note that the generalisability of* $\mathbb{H}_{sh}$ *(Table 4.6), despite it having the volume 19 order of magnitudes bigger, is only 3% greater than* $\mathbb{H}_{word}$. 82
4.11 *Total volume and percentage of the training embedding space covered by our best hyper-rectangles. The total volume of the training embedding space for RUAR is* $6.14e-5$, *and for Medical is* $1.43e-5$. 82
4.12 *Accuracy of the robustly trained DNNs on the RUAR and the Medical datasets.* $\mathscr{Y}$ *stands for either RUAR or Medical depending on the column.* 84

4.13 *Verifiability of the robustly trained DNNs on the RUAR and the Medical datasets, for a selection of semantic subspaces; using the ERAN verifier.* 84
4.14 *Verifiability of the DNNs trained for robustness on the RUAR and the Medical datasets, for a selection of semantic subspaces; using the Marabou verifier.* . . 85
4.15 *Accuracy of the DNNs trained adversarially on the RUAR and the Medical datasets.* 85
4.16 *Verifiability of the DNNs trained adversarially on the RUAR and the Medical datasets, for a selection of geometric subspaces; using the ERAN verifier.* . . . 85
4.17 *Comparison of different subspace shapes for verification and verifiers for* N_{pj-adv} *and* $\mathbb{H}_{pj}$ *on the Medical dataset.* 86
4.18 *Section 4.4 NLP verification pipeline setup, implemented using ANTONIO. Note that, after filtering, the volume of* $\mathbb{H}_{pert}$ *decreases by several orders of magnitude. Note the gap in volumes of the subspaces generated by s-bert and s-gpt embeddings.* 92
4.19 *Performance of the models on the test/perturbation set. The average standard deviation is* 0.0049. 94
4.20 *Annotation instructions for manual estimation of the perturbation validity.* . . . 96
4.21 *Semantic similarity results of the manual evaluation for annotators A1 and A2.* . 97
4.22 *Grammaticality results of the manual evaluation for annotators A1 and A2.* . . . 97
4.23 *Label consistency results of the manual evaluation for annotators A1 and A2.* . . 97
4.24 *Number of perturbations kept for each model after filtering with cosine similarity > 0.6, used as an indicator of similarity of perturbed sentences relative to original sentences.* 99
4.25 *Performance of the models on the test/perturbation set, after filtering. The average standard deviation is* 0.0049. 99
4.26 *ROUGE-N scores comparing the original samples with Vicuna perturbations (of the positive class) for lexical overlap.* 101
4.27 *ROUGE-N scores comparing the original samples with Vicuna perturbations (of the positive class) for syntax overlap.* 101
4.28 *Verifiability, generalisability and embedding error of the baseline and the robustly (adversarially) trained DNNs on the RUAR and the Medical datasets, for* $\mathbb{H}_{pert^{\diamond}}$ *(*N_{base} *and* $N_{pert^{\diamond}}$*) and* $\mathbb{H}_{pert}$ *(*N_{pert}*); for Marabou verifier.* 104

5.1 *Categorisation of key NIDS features used in the training-verification pipeline.* 114
5.2 *Alphabetical index of constraint building properties with cross–references to their formal definitions.* 122
5.3 *Summary of the eight global NIDS verification properties.* 124
5.4 *F1 scores for NIDS models across datasets and attack types.* 128
5.5 *Verifiability of the networks over our eight properties.* 129
5.6 *Verification of the* `BenignTraffic` *specification across model sizes.* 130
5.7 *Coverage of specifications over labelled dataset flows.* ***Malicious**** *denotes flows matched by the verified subset of the full malicious specifications.* 132

List of Figures

1.1 *Left: clean stop sign image [7]. Right: stop sign with simulated dirt or sticker noise. A robust model should classify both as 'stop'.* 4
1.2 *Continuous Verification Cycle [8]: a closed loop linking training, verification, and re-training based on a shared property.* 7

2.1 *(a) A verifiable but poorly generalisable ε-ball; (b) convex hull; (c) hyper-rectangle; (d) rotated hyper-rectangle. Red = training embeddings; turquoise = test embeddings of the same class.* 23
2.2 *A VNNLib specification of ACASXu Property 3: COC must not be chosen when the intruder is directly ahead and moving toward the ownship.* 24
2.3 *A Vehicle specification for ACASXu Property 3: under approach conditions, the network must not advise COC as the minimal output.* 25
2.4 *Visualisation of the NLP verification pipeline followed in our approach.* 43

3.1 *Constraint security of SR and LR-trained networks against PGD attacks using different robustness definitions. ε denotes attack strength.* 51
3.2 *Experiments that show how adversarial training, training with data augmentation, and training with constraint loss affect standard and classification robustness of networks; ε measures the attack strength.* 52
3.3 *Images from MNIST: (left) high confidence; (right) low confidence.* 53
3.4 *Experiments that show how different choices of a constraint loss affect standard robustness of neural networks.* 54
3.5 *An illustrative Vehicle specification for CR(ε). This was not used in practice but shows how such a property could be encoded formally.* 55

4.1 *Visualisation of the NLP verification pipeline followed in our approach.* 63
4.2 *An example of hyper-rectangle drawn around all points of the same class (a), shrunk hyper-rectangle $\mathbb{H}_{sh}$ that is obtained by excluding all points from the opposite class (b) and clustered hyper-rectangles (c) in 2-dimensions. The red dots represent sentences in the embedding space of one class, while the blue dots are embedded sentences that do not belong to that class.* 69
4.3 *An illustrative Vehicle specification for NLP robustness. While not used for verification here, it formalises the geometric subspace constraint described in Section 4.2.4.* 74
4.4 *Tool ANTONIO that implements a modular approach to the NLP verification pipeline used in this work.* 87
4.5 *Zero-shot prompts with 2 basic examples from the R-U-A-Robot dataset. Answers from `vicuna-13b` are given in italics. A1 and A2 represent different answers to the same prompt, illustrating a lack of consistency in the output.* 91
4.6 *Analysis of some common issues found in the `vicuna-13b` generated perturbations.* 102
4.7 *In this figure, we show how a prepended, semantically informed verified filter added to an NLP system (here, an LLM), can check that safety-critical queries are handled responsibly, e.g. by redirecting the query to a tightly controlled rule-based system instead of a stochastic LLM.* 110

5.1 *Adapted training-verification pipeline for NIDS.* 117

5.2 *Vehicle encodings for the eight global properties.* 124
5.3 *Constraint Security (CSec) under local adversarial attacks, plotted against perturbation size* ε. 131

Glossary

AI Artificial Intelligence.

BaB Branch and Bound.

CAcc Constraint Accuracy.

CI Classification Invariance.

CNN Convolutional Neural Network.

CR Classification Robustness.

CSat Constraint Satisfaction.

CSec Constraint Security.

CV Computer Vision.

DL Deep Learning.

DNN Deep Neural Network.

DoS Denial of Service.

FGSM Fast Gradient Sign Method.

FTP File Transfer Protocol.

GRU Gated Recurrent Unit.

GTSRB German Traffic Sign Recognition Benchmark.

HTTP Hypertext Transfer Protocol.

IAA Inter Annotator Agreement.

IAT Inter Arrival Time.

IBP Interval Bound Propagation.

ICC Intraclass Correlation Coefficient.

IDS Intrusion Detection Systems.

IP Internet Protocol.

LLM Large Language Model.

LR Lipschitz Robustness.

LSTM Long Short Term Memory.

MILP Mixed Integer Linear Programming.

ML Machine Learning.

NIDS Network Intrusion Detection Systems.

NLP Natural Language Processing.

NN Neural Network.

PGD Projected Gradient Descent.

POS Parts of Speech.

RNN Recurrent Neural Network.

RUAR R U A Robot.

SCR Strong Classification Robustness.

SDG Stochastic Gradient Descent.

SMT Satisfiability Modulo Theories.

SQl Structured Query Language.

SR Standard Robustness.

SSH Secure Shell.

TCP Transmission Control Protocol.

UDP User Datagram Protocol.

URL Uniform Resource Locator.

VNN-COMP International Verification of Neural Networks Competition.

List of Publications

Marco Casadio et al. “Neural network robustness as a verification property: a principled case study”. In: *International conference on computer aided verification*. Springer. 2022, pp. 219–231.

Marco Casadio et al. “NLP verification: towards a general methodology for certifying robustness”. In: *European Journal of Applied Mathematics* (2025), pp. 1–58.

Marco Casadio et al. “ANTONIO: Towards a Systematic Method for Generating NLP Benchmarks for Verification”. In: *Proceedings of the 6th Workshop on Formal*. Vol. 16. 2023, pp. 59–70.

Robert Flood et al. “Formally Verifying Robustness and Generalisation of Network Intrusion Detection Models”. In: *Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing*. 2025, pp. 1867–1876.

Matthew L Daggitt et al. “The Vehicle Tutorial: Neural Network Verification with Vehicle.” In: *FoMLAS@ CAV*. 2023, pp. 1–5.

Robert Flood et al. “Generating Traffic-Level Adversarial Examples from Feature-Level Specifications”. In: *European Symposium on Research in Computer Security*. Springer. 2024, pp. 118–127.

Colin Kessler et al. “Neural Network Verification for Gliding Drone Control: A Case Study”. In: *SAIV@ CAV*. 2025.

Thomas Flinkow et al. “A Generalised Framework for Property-Driven Machine Learning”. In: *arXiv preprint arXiv:2505.00466* (2025).

Chapter 1

Introduction

Contents

1.1 Motivation and Scope 3

1.2 Methodological Framework 5

1.3 Thesis Contributions 7

1.4 Limitations 9

1.5 Contribution Declaration 9

1.6 List of Additional Papers 10

Machine learning (ML) systems, and in particular deep neural networks (DNNs), have achieved remarkable success in domains such as image recognition, natural language processing, and cyber-security. As their adoption expands into safety-critical applications—autonomous driving, healthcare diagnostics, network intrusion detection—so too does the need for guarantees about their behaviour. Small changes to input data, whether visual noise, synonymous sentence rephrasings, or altered network traffic features, can trigger large and unpredictable changes in model output. These failures highlight a core vulnerability: neural networks, while accurate, are often unreliable under distribution shifts or perturbations [1]. Since around 2014, both the ML and formal verification communities have studied this challenge extensively, developing complementary approaches [2, 3, 4].

This thesis studies how to apply verification and ML methods in order to improve the behaviour of DNNs under such perturbations. Our focus is on the integration of ML methods and formal verification of DNN properties: mathematical guarantees that a model behaves consistently within well-defined perturbation regions or subspaces of the input space. These regions include, for instance, ε-balls (which contain all points within a distance ε of an input) or *hyper-rectangles* (which constrain each input feature independently). This involves identifying a target *property*, aligning the model's training process with that property, and verifying its satisfaction using automated tools. We

develop a unified methodology for this process and apply it across three representative domains: computer vision, natural language processing (NLP), and network intrusion detection systems (NIDS). In this thesis, we study two kinds of properties: *local properties*, defined as small neighbourhoods around a given point (specifically ε-balls $\mathbb{B}(\hat{x}, \varepsilon)$ with a uniform distance ε from $\hat{x}$), and *global properties*, defined as axis-aligned hyper-rectangles (by giving intervals independently for each dimension). We reserve the term *local robustness* for robustness properties stated on ε-balls around a reference input, and use *global properties* for statements over arbitrary axis-aligned hyper-rectangles. Formal definitions appear later in Chapters 3 and 4.

A key theme throughout this thesis is the study of methods that link symbolic logical constraints on inputs (verification constraints) with the geometry of input spaces that can be explored in both training and verification. Formally, a deep neural network can be seen as a function $N : \mathbb{R}^m \to \mathbb{R}^n$, which operates on continuous vector inputs. However, many real-world domains provide inputs that are discrete or symbolic in nature, such as words in natural language or protocol fields in network traffic. To apply neural networks in these settings, one must use an *embedding function* that maps these symbolic inputs into continuous vector spaces where DNNs operate. These embeddings may be hand-crafted (e.g., one-hot encodings), normalised numeric features (e.g., pixels normalised between 0 and 1), or learned representations (e.g., transformer encoders). While these transformations enable learning, they also reshape the structure of the input space. In computer vision, the mapping from raw images to embedding space (e.g., via normalisation) tends to preserve semantic meaning. This is because continuous domains like image space or sensor data already possess a natural geometric structure, which aligns well with the vector space operations of DNNs. As a result, small pixel-level perturbations often yield perceptually similar inputs and correspond to nearby points in the embedding space. By contrast, in NLP, this is not guaranteed. Two semantically similar sentences like:

> "Are you a robot?" vs. "Am I speaking to an AI assistant?"

may be mapped far apart in the embedding space. Meanwhile, semantically unrelated sentences like:

> "You're not human, right?" vs. "Do you know the movie 'Are you a robot?'?"

may appear deceptively close. This leads to what we call the *embedding gap*: a mismatch between the geometry used by DNNs and the underlying semantics of the domain.

In NLP, this issue is compounded by the fact that embeddings are not invertible: most vectors in the continuous space do not correspond to valid sentences. This makes it difficult to reason directly about model behaviour in semantic terms. The gap complicates robustness analysis, as distances in the embedding space do not reliably reflect semantic similarity.

We argue that hyper-rectangles—defined geometrically, either directly via ε-balls or through sets of generated related sentences—offer a unified formalism for addressing these challenges. If a property can be expressed as a hyper-rectangle, then it can be trained for, attacked, and verified using standard tools. The chapters that follow explore how far this idea generalises.

We begin by identifying two persistent obstacles in the literature: (i) low verifiability of real-world models; and (ii) a mismatch between semantic intent and geometric representation (the embedding gap). These are the core problems this thesis addresses.

1.1 Motivation and Scope

This thesis is motivated by two key limitations in the current state of DNN verification:

1. **Low verifiability.** Despite progress in verification tools, many models remain difficult to verify in practice [5]. Here, by "verifiability" we mean the extent to which meaningful properties of a model can be formally defined and checked using existing verification frameworks. Many real-world models lack verifiability in this sense: either the intended property is hard to express in the tool's input language (for example, it would require many disjoint regions or relational constraints), or the resulting queries are too large or imprecise to be proved within reasonable resources, especially in high-dimensional or semantically complex domains.

2. **The embedding gap.** In domains where inputs are symbolic or structured (e.g., NLP), semantic meaning must be approximated through learned embeddings into real vector spaces. These representations are often non-invertible and misaligned with human judgments, making robustness properties difficult to define geometrically [6].

Verification challenges manifest differently depending on the nature of the data, the

properties of interest (e.g., local robustness vs. global properties), and the kinds of perturbations considered. The following examples illustrate this diversity:

Example 1 (Computer Vision) *Suppose a neural network is trained to classify road signs. A local robustness property can be stated as: for any stop-sign image $\hat{x}$, the predicted class must be invariant throughout an ε-ball of radius ε in pixel space around $\hat{x}$. This property is definable (it can be written geometrically as bounds on inputs) and, in principle, checkable with standard verifiers. In practice, verifiability is hard: the ε-ball lies in a very high-dimensional space, so verification queries become large. Moreover, small mismatches between the training objective and the verification property can make proofs fail. There is also a mild embedding gap even in vision. Reasoning takes place in normalised pixel space rather than strictly on the manifold of valid images; large portions of an ε-ball can include off-manifold or perceptually altered images that no longer look like a clean stop sign. The verifier must therefore certify a geometric region that is broader than the set of semantically valid variants, complicating both specification (what region do we actually intend?) and checkability (bigger regions are harder to prove).*

Figure 1.1: *Left: clean stop sign image [7]. Right: stop sign with simulated dirt or sticker noise. A robust model should classify both as 'stop'.*

Example 2 (Natural Language Processing) *Consider an AI chatbot that is legally required to disclose its non-human identity when prompted. A user may ask: "Are you a robot?", but they might also say "Am I chatting with a real person?" or "You're not human, right?". The semantic content remains the same, but the sentence forms vary, and the system must respond consistently to these semantically equivalent queries. To*

make this verifiable, we must define a property over a subspace that collects semantically equivalent inputs. Operationally, we embed sentences and build a hyper-rectangle that encloses an input sentence and its paraphrases, and the property states that the classifier's output is constant throughout that region. This exposes two verifiability hurdles: (i) definability—because embeddings are non-invertible, many points in the hyper-rectangle do not correspond to valid sentences, and naive geometric regions may fail to capture the intended semantics; and (ii) checkability—high-dimensional regions can be difficult for verifiers to prove. Both issues stem from the embedding gap between geometry and linguistic meaning.

Example 3 (Network Intrusion Detection) *In network security, DNNs may be deployed to classify network traffic as benign or malicious. Certain rules—e.g., "no packet with protocol X and port Y should be marked benign"—must hold universally. These are global, rule-based properties defined over mixed discrete and continuous input fields and can often be translated into one or more hyper-rectangles for training and verification. Verifiability then hinges on checkability: unions over discrete cases may explode combinatorially, large rule regions produce loose bounds, and verifiers can struggle at this scale. Moreover, engineered feature encodings (e.g., one-hot for categorical fields) can distort semantic structure, aggravating the specification–geometry mismatch. Thus, while the property is definable, proving it at realistic scales can still be difficult.*

These three cases span different types of data (continuous, discrete, mixed), different kinds of properties (local robustness vs. global properties), and different representational challenges (embedding gap, semantic ambiguity, engineered feature spaces). They illustrate the need for a unifying methodology that can adapt to different settings while addressing the core problems of verifiability and representational mismatch.

1.2 Methodological Framework

This thesis develops a principled methodology for training and verifying DNNs with respect to formally defined properties over geometrically specified input regions. These properties may describe local robustness (e.g., classification consistency under small perturbations) or global properties (e.g., "no traffic with source port 22 should be classified as benign"). The unifying feature is that all properties are defined over *subspaces* of

the input domain — and specifically, over regions that can be described using *hyper-rectangles*.

A hyper-rectangle in n dimensions is a product of closed intervals — one for each input feature — and corresponds exactly to a set of independent linear constraints. These are conjunctions of lower and upper bounds on each input variable, of the form $\alpha_i \leq x_i \leq \beta_i$. For example, a region can be defined by:

$$0 \leq x_1 \leq 1, \quad 4 \leq x_2 \leq 5, \quad 3 \leq x_3 \leq 8, \quad 1 \leq x_4 \leq 7$$

forms a 4-dimensional hyper-rectangle. These shapes are expressive enough to encode both small perturbation regions (e.g., ε-balls) and large, rule-based domains (e.g., input combinations that meet domain-specific conditions). Crucially, hyper-rectangles are also tractable: they align with the input abstractions used by most formal verifiers, and can be used to structure both training data and verification queries.

This framing motivates a unified training-verification pipeline, where the same subspaces — described as hyper-rectangles — guide both the training process and the verification task. Rather than treating training and verification as separate phases, we advocate aligning them around a shared *specification*: a formally defined logical property tied to specific input regions via hyper-rectangles.

The methodological pipeline, summarised in Figure 1.2, consists of the following stages:

1. **Property Definition**: Specify the property of interest (e.g., classification robustness, Lipschitz robustness, global rules) as a formal constraint over inputs and outputs.

2. **Training Aligned to Property**: Apply training methods (e.g., data augmentation, (customised)-adversarial training, constraint-based loss) that are designed to optimise for the chosen property.

3. **Verification and Evaluation**: Use a formal verifier to check whether the trained model satisfies the property over specified regions of the input space.

4. **Retraining (if needed)**: If verification fails, retrain the model using insights from the failure (e.g., by adjusting the training set or loss function).

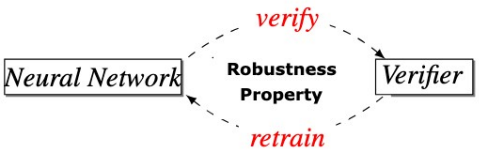


Figure 1.2: *Continuous Verification Cycle [8]: a closed loop linking training, verification, and retraining based on a shared property.*

This continuous verification loop applies across all domains studied in the thesis, but must be adapted to domain-specific representations and property definitions:

- In computer vision, robustness is defined geometrically over ε-balls around inputs in pixel space. The embedding is trivial (e.g., input normalisation), and verification methods are mature and efficient.
- In NLP, inputs are discrete and embedded into high-dimensional vector spaces. We define classification invariance over *hyper-rectangles* in the embedding space (see Definition 14, Chapter 4), constructed from semantically similar sentences. We also adapt adversarial training to operate over these subspaces.
- In NIDS, inputs include both discrete and continuous features. Properties are defined globally, through logical rules spanning multiple features. These global properties are expressed as hyper-rectangles in structured feature space, allowing for both rule encoding and verifiability.

Throughout, we clarify the relationships between properties formally, as a training objective, and as a verification goal — helping practitioners identify mismatches and improve DNN guarantees.

1.3 Thesis Contributions

This thesis develops a unified methodology for training and verifying DNNs with respect to formally defined properties over geometrically defined subspaces of the input. These properties include both traditional robustness notions (e.g., consistency under small perturbations) and global, rule-based specifications. Our approach is grounded in a shared representation — the *hyper-rectangle* — which enables us to apply the same geometric framework across domains, and tackle the key challenges of low verifiability and

representational mismatch introduced in Section 1.1. We focus on the three representative domains previously declared and contribute to each as follows:

Computer Vision (Chapter 3). We identify and compare four robustness properties used in verification literature: classification robustness (CR), strong classification robustness (SCR), standard robustness (SR), and Lipschitz robustness (LR). These definitions are not equivalent: some are logically stronger (e.g., LR implies SR), while others (e.g., SR vs. SCR) are incomparable. We show how different robust training methods correspond to these four properties, and that mismatches between the training objective and the verification property often lead to reduced verifiability. We also introduce a shared evaluation framework based on three metrics (constraint satisfaction, constraint security, and constraint accuracy) and demonstrate that different properties lead to distinct robustness guarantees under these metrics. Overall, this chapter clarifies conceptual confusion in the literature and establishes a foundation for implementing principles of continuous verification across different domains in subsequent chapters.

Natural Language Processing (Chapter 4). We address the unique challenges of verifying robustness in NLP, where the discrete and semantic nature of the inputs exacerbates the embedding gap and undermines verifiability. Our contribution is twofold: (1) we define a method for constructing *semantic subspaces* in the embedding space, where each subspace is formed by embedding a sentence along with its semantically related variants, and enclosing them within a hyper-rectangle; (2) we design a robust training procedure based on adversarial Projected Gradient Descent (PGD) [9], and adapt it to operate over such hyper-rectangles in the embedding space. This allows the model to learn to behave consistently across semantically equivalent inputs. Our verification pipeline operates entirely over continuous embeddings but is guided by semantic information from language. This targets the gap between geometric verification and linguistic meaning. We also implement a tool, ANTONIO, which automates the generation of semantic perturbation sets and corresponding embedding subspaces, and uses them for training and verification. Our experiments show that this semantically informed training substantially improves verifiability.

Network Intrusion Detection (Chapter 5). We extend our verification pipeline to NIDS, where inputs are structured vectors comprising both discrete and continuous fea-

tures. In this domain, we specify global classification invariance as rule-based hyper-rectangles over interpretable features (e.g., "traffic from port 22 should never be marked benign"). These regions support both verification and property-driven training, where the model is optimised to satisfy the logical constraint throughout each region. Our results show that, despite differences in data type and semantics, the same specification-first training–verification methodology applies across domains (computer vision, NLP, NIDS). Chapters 4 and 5 both use global classification invariance (NLP via paraphrase-derived boxes; NIDS via domain rules), with minimal changes to the pipeline.

Together, these contributions advance DNN verification across multiple domains by aligning training objectives with formal properties.

1.4 Limitations

While this thesis offers a general and cross-domain methodology, it does make certain assumptions about the kinds of properties considered and the form of robustness targeted. Although our method is extremely general, there is one more generalisation which could be done, which was investigated with the author of Flinkow et al. [10], but due to the collaborative nature of the work is not put as an independent chapter in this thesis. It is important to note that this thesis focuses on class robustness, that is, the ability of a model to maintain consistent predictions under constrained input perturbations. This is only one facet of a broader robustness landscape. For example, recent work by Thomas Flinkow [11] suggests that certain properties may require more elastic or nuanced interpretations of model output, which go beyond simple class invariance. Such directions are not addressed here, but they highlight the complexity and richness of the space in which this work sits.

1.5 Contribution Declaration

The following papers form the basis for Chapters 3, 4, and 5. In all the papers below, the PhD candidate was the first author. The significant contributions by other authors will be explicitly listed below

- Marco Casadio, Ekaterina Komendantskaya, Matthew L. Daggitt, Wen Kokke, Guy Katz, Guy Amir, Idan Refaeli: Neural Network Robustness as a Verification Property: A Principled Case Study. CAV 2022: 219-231

Guy Katz, Guy Amir and Idan Refaeli performed the verification. Matthew Daggitt provided the relationships between robustness properties.

- Marco Casadio, Tanvi Dinkar, Ekaterina Komendantskaya, Luca Arnaboldi, Matthew L. Daggitt, Omri Isac, Guy Katz, Verena Rieser, Oliver Lemon: NLP verification: towards a general methodology for certifying robustness. EJAM 2025: 1-58
  Tanvi Dinkar provided the ROUGE-N results and language analysis. Luca Arnaboldi provided the statistical analysis of IAA and ICC. Omri Isac provided the Marabou results.

- Marco Casadio, Luca Arnaboldi, Matthew L. Daggitt, Omri Isac, Tanvi Dinkar, Daniel Kienitz, Verena Rieser, Ekaterina Komendantskaya: ANTONIO: Towards a Systematic Method of Generating NLP Benchmarks for Verification. FoMLAS@CAV 2023: 59-70
  Omri Isac provided the Marabou results. Daniel Kienitz provided the data rotation.

- Robert Flood, Marco Casadio, David Aspinall, Ekaterina Komendantskaya: Formally Verifying Robustness and Generalisation of Network Intrusion Detection Models. SAC 2025: 1867-1876
  Robert Flood provided datasets, feature engineering, pre-processing and adversarial examples.

1.6 List of Additional Papers

The author also contributed to the following peer-reviewed publications not included in this thesis:

- Matthew L. Daggitt, Wen Kokke, Ekaterina Komendantskaya, Robert Atkey, Luca Arnaboldi, Natalia Slusarz, Marco Casadio, Ben Coke, Jeonghyeon Lee: The Vehicle Tutorial: Neural Network Verification with Vehicle. FoMLAS@CAV 2023: 1-5

- Robert Flood, Marco Casadio, David Aspinall, Ekaterina Komendantskaya: Generating Traffic-Level Adversarial Examples from Feature-Level Specifications. ESORICS Workshops 2024: 118-127

- Colin Kessler, Ekaterina Komendantskaya, Marco Casadio, Ignazio Maria Viola,

Albaraa Ammar Othman, Alistair Malhotra, Robbie McPherson: Neural Network Verification for Gliding Drone Control: A Case Study. SAIV 2025: Proceedings

- Thomas Flinkow, Marco Casadio, Colin Kessler, Rosemary Monahan, Ekaterina Komendantskaya: A Generalised Framework for Property-Driven Machine Learning. arXiv:2505.00466

Chapter 2

Background & Literature Review

Contents

2.1 Background 13

2.1.1 Deep Neural Networks and Data Representations 13

2.1.2 Examples of DNN Applications 14

2.1.3 Adversarial Examples 16

2.1.4 Formal Definitions of Robustness and Training Techniques . . 18

2.1.5 Geometric Analysis of Embedding Spaces 21

2.1.6 Specification Languages for Neural Network Verification . . . 23

2.1.7 Local vs Global Properties 26

2.2 Literature Review 27

2.2.1 Robust Training 27

2.2.2 DNN Verification 29

2.2.3 Robustness and Verification in Computer Vision 32

2.2.4 NLP Robustness 33

2.2.5 Previous NLP Verification Approaches 37

2.2.6 Datasets Used in NLP Verification 39

2.2.7 NIDS Robustness and Verification 41

2.2.8 A Systematic View of Training-Verification Pipelines 43

In this chapter, we present the relevant background knowledge and review the literature related to the contributions of this thesis. We begin in Section 2.1 by outlining the foundational concepts, starting with the definition of Deep Neural Networks (DNNs) and progressing through key topics such as different forms of input data, adversarial attacks on ML models, an analysis on geometric inputs representations, and different specification languages for NN verification. Following this, Section 2.2 provides a comprehensive literature review, beginning with an exploration of robust training and standard DNN

verification techniques, followed by robustness and verification in Natural Language Processing (NLP) and in Network Intrusion Detection System (NIDS) and concluding with presenting a systematic approach of DNN robustness and verification pipelines.

2.1 Background

2.1.1 Deep Neural Networks and Data Representations

A deep neural network is a function $N : \mathbb{R}^m \to \mathbb{R}^n$ that maps an input space $\mathbb{R}^m$ to an output space $\mathbb{R}^n$. We will use $\hat{x} \in \mathbb{R}^m$ to denote an input vector and $y \in \mathbb{R}^n$ an output vector. In this thesis, a DNN N is typically used to model classifiers that assign inputs to one of n classes.

A DNN consists of a composition of layers, each parameterised by a weight matrix and bias vector, and applying a non-linear transformation to its input

Definition 1 (Neural Network Layer) *A DNN is a composition of layers $L_1 \circ \ldots \circ L_L$, where each layer L^l is defined as:*

$$h^l = \sigma(W^l h^{l-1} + b^l)$$

with $h^0 = \hat{x}$ as the input to the network and h^L as the final output. Here, W^l is the weight matrix, b^l is the bias vector, and σ is an activation function (typically ReLU [12] in this thesis).

Each layer transforms its input using affine operations, building internal representations that support the optimisation task at hand. While it is common to interpret deeper layers as learning higher-level abstractions, such interpretations are heuristic and often domain-specific.

Training a DNN involves learning parameters $\theta = \{W^l, b^l\}_{l=1}^L$ that minimise a loss function $\mathscr{L}$, measuring the discrepancy between predictions $N(\mathbb{X})$ and ground-truth labels $\mathbb{Y}$, where $\mathbb{X} \subset \mathbb{R}^m$ is a set of inputs and $\mathbb{Y} \subset \mathbb{R}^n$ its corresponding set of outputs.

The choice of loss function depends on the task. For classification tasks, the cross-entropy loss is standard:

$$\mathscr{L} = -\sum_{i=1}^{n} \mathbb{Y}_i \log N(\mathbb{X})_i$$

where $\mathbb{Y}_i$ is the true class label (typically one-hot encoded), and $N(\mathbb{X})_i$ is the predicted

probability for class i.

For a regression task, a common loss function is the mean squared error (MSE), defined as:

$$\mathscr{L} = \frac{1}{n}\sum_{i=1}^{n}(\mathbb{Y}_i - N(\mathbb{X})_i)^2$$

which penalises large deviations between predictions and ground-truth values.

The network parameters are then adjusted using gradient-based methods, such as Stochastic Gradient Descent (SGD) [13] or adaptive optimisers like Adam [14], using updates of the form:

$$\theta^{(t+1)} = \theta^{(t)} - \eta\nabla_\theta\mathscr{L}$$

where η is the learning rate, and $\nabla_\theta\mathscr{L}$ denotes the gradient of the loss function $\mathscr{L}$ with respect to the network parameters θ, i.e., the vector of partial derivatives $\left[\frac{\partial\mathscr{L}}{\partial\theta_1}, \frac{\partial\mathscr{L}}{\partial\theta_2}, \ldots\right]$ indicating how changes in parameters affect the loss.

Domains and Representations. The input domain $\mathscr{D}$ varies across applications. In many real-world settings, especially when inputs are symbolic or structured, a domain-specific *embedding function* $E : \mathscr{D} \to \mathbb{R}^m$ is used to map inputs into the real vector space that the DNN operates over. The following section outlines the datasets and representations used in three application domains: computer vision, natural language processing, and network intrusion detection systems.

2.1.2 Examples of DNN Applications

Computer Vision. In computer vision, the input domain is a space of real-valued image tensors. We use two image classification datasets:

- **Fashion-MNIST** [15]: 28×28 greyscale images from 10 fashion categories.
- **GTSRB** [7]: German traffic sign images, preprocessed to 48×48 greyscale with 10 selected classes.

In both cases, the embedding is lightweight: pixel values are flattened into vectors and normalised. The classifier N operates directly on $\mathbb{R}^m$ and maps to $\mathbb{R}^n$.

Natural Language Processing. In NLP, we consider the set of all strings $\mathbb{S}$, where an NLP dataset $\mathscr{Y} \subset \mathbb{S}$ is a set of sentences $s_1, \ldots, s_q$ written in natural language. The embedding function E then maps a string in $\mathbb{S}$ to a vector in $\mathbb{R}^m$. Ideally, E should reflect

the semantic similarities between sentences in $\mathbb{S}$, i.e. the more semantically similar two sentences s_i and s_j are, the closer the distance between $E(s_i)$ and $E(s_j)$ should be in $\mathbb{R}^m$. Of course, defining semantic similarity in precise terms may not be tractable (the number of unseen sentences may be infinite, the similarity may be subjective and/or depend on the context). This is why, the state-of-the-art NLP relies on machine learning methods to capture the notion of semantic similarity approximately. Currently, the most common approach to obtain an embedding function E is by training *transformers* [16, 17]. Transformers are a type of DNNs that can be trained to map sequential data into real vector spaces and are capable of handling variable-length input sequences. They can also be used for other tasks, such as classification or sentence generation, but in those cases, too, training happens at the level of embedding spaces. In this work, a transformer is used as a function $E : \mathbb{S} \to \mathbb{R}^m$ for some given m. The key feature of the transformer is the "self-attention mechanism", which allows the network to weigh the importance of different elements in the input sequence when making predictions, rather than relying solely on the order of elements in the sequence. This makes them good at learning to associate semantically similar words or sentences. We initially use the transformer Sentence-BERT [17] and later add Sentence-GPT [18] to embed sentences. Unfortunately, the relation between the embedding space and the NLP dataset is not bijective: i.e. each sentence is mapped into the embedding space, but not every point in the embedding space has a corresponding sentence. This problem is well-known in NLP literature [6] and, as shown in this thesis, is one of the reasons why verification of NLP is tricky. Given an NLP dataset $\mathscr{Y}$ that should be classified into n classes, the standard approach is to construct a function $N : \mathbb{R}^m \to \mathbb{R}^n$ that maps the embedded inputs to the classes. In order to do that, a domain specific classifier N is trained on the embeddings $E(\mathscr{Y})$ and the final system will then be the composition of the two subsystems, i.e. $N \circ E$.

We use two safety-critical binary classification datasets:

- **R-U-A-Robot** [19]: 6,800 variations of identity-disclosure prompts (e.g. "Are you human?").
- **Medical Safety** [20]: 2,917 Reddit questions, classified as medical or non-medical.

Network Intrusion Detection. NIDS models operate on structured tabular data, drawn from domains like packet-level features or connection logs. The datasets used are:

- **CIC IDS 2017** [21] and **CIC IDS 2018** [22]: Network flow data annotated as

benign or malicious.

Each data point contains both continuous features (e.g. packet length) and categorical features (e.g. protocol type). These are normalised or encoded (e.g. one-hot) into a vector in $\mathbb{R}^m$, yielding an implicit embedding function $E : \mathscr{D} \to \mathbb{R}^m$. Unlike NLP, these encodings are designed for fidelity rather than semantic structure.

Across these domains, the embedding function E mediates between symbolic or structured data and the continuous vector space $\mathbb{R}^m$ used by DNNs. The form of E—whether lightweight (computer vision), learned (NLP), or engineered (NIDS)—shapes how we define and verify robustness in each setting. These distinctions—between continuous, discrete, and structured data, and their associated embeddings—form the basis of the three-axis framework introduced in Chapter 1, and guide the methodological pipeline throughout the thesis.

2.1.3 Adversarial Examples

Adversarial examples are inputs to machine learning models that have been perturbed in a way that is often imperceptible to humans but cause the model to make incorrect predictions with high confidence. These perturbations are typically crafted by adding small, carefully calculated noise to the original input data, exploiting the model's learned decision boundaries. The concept of adversarial examples was first introduced in the context of image classification, where subtle modifications to an image can lead a DNN to misclassify it entirely [2, 1].

Formally, given a classifier $N : \mathbb{R}^m \to \mathbb{R}^n$ and an input $\hat{x} \in \mathbb{R}^m$ with its true label $y \in 1, \ldots, n$, an *adversarial example* x is a perturbed version of $\hat{x}$ such that:

$$\arg\max_i N(x)_i \neq \arg\max_i N(\hat{x})_i,$$

where x is constrained to remain within a small perturbation region around $\hat{x}$, typically defined by a distance metric $d(\hat{x}, x)$, and $\arg\max_i N(x)_i$ returns the index of the output class with the highest score, i.e., the predicted label of input x. A common choice is the ℓ_p norm:

$$d(\hat{x}, x) = |x - \hat{x}|_p \leq \varepsilon,$$

where $\varepsilon > 0$ controls the magnitude of the perturbation. Two widely used values of p are:

- $p = \infty$, which defines an ℓ_∞-bounded adversarial example, restricting each individ-

ual feature perturbation to at most ε, resulting in an ε-cube around $\hat{x}$.

- $p = 2$, which defines an ℓ_2-bounded adversarial example, where the perturbation is constrained to lie within an ε-ball around $\hat{x}$, allowing small modifications across all features rather than restricting each component individually.

Small values of ε ensure that x remains visually or structurally similar to $\hat{x}$ while still causing misclassification.

The phenomenon of adversarial examples highlights a significant vulnerability in DNNs: despite their remarkable ability to generalise across large datasets, they can be easily fooled by small alterations that are virtually indistinguishable from the original input. This vulnerability poses a substantial threat in various application areas, from autonomous driving and facial recognition to NLP and security systems like NIDS [23, 24].

Generation of Adversarial Examples

Adversarial examples are typically generated using attack algorithms that aim to maximize the error of the model by introducing small perturbations into the input. Two of the most well-known methods for generating adversarial examples are the Fast Gradient Sign Method (FGSM) and Projected Gradient Descent (PGD).

- *FGSM* [1] is a simple and efficient method for crafting adversarial examples. It works by calculating the gradient of the loss function with respect to the input and then adding a small perturbation in the direction of the gradient. The amount of perturbation is controlled by a parameter $\varepsilon \in \mathbb{R}$, which determines the magnitude of the noise added to the input. Formally, for an input $\hat{x} \in \mathbb{R}^m$ and a loss function $\mathscr{L}$, the adversarial example x is computed as:

$$x = \hat{x} + \varepsilon \cdot \text{sign}(\nabla_{\hat{x}} \mathscr{L}(\hat{x}, y))$$

  where $\nabla_{\hat{x}} \mathscr{L}(\hat{x}, y)$ is the gradient of the loss with respect to the input, and $\text{sign}(\cdot)$ denotes the element-wise sign function, returning $+1$ for positive entries, -1 for negative entries, and 0 for zeros.

- *PGD* [9] is an iterative extension of FGSM that performs multiple steps of gradient descent to create more effective adversarial examples. In each iteration, a small

perturbation is added to the input, and the resulting adversarial example is projected back into a predefined subspace, usually ensuring that the perturbation remains imperceptible. The traditional PGD algorithm is defined as follows: given a loss function $\mathscr{L}$, a step size $\gamma \in \mathbb{R}$, and an input $\hat{x} \in \mathbb{R}^m$, the output of the PGD algorithm $x(l)$ after l iterations is:

$$\begin{aligned} x(0) &= \hat{x} \\ x(t+1) &= \text{proj}_{\mathscr{S}}\Big[x(t) + \gamma \cdot \text{sign}(\nabla_{x(t)}\mathscr{L}(x(t), y))\Big] \end{aligned}$$

where $\text{proj}_{\mathscr{S}}$ represents the projection back into a predefined subspace $\mathscr{S}$, which is often an ε-ball around the original input to limit the perturbation. This method is highly effective for adversarial training, where models are trained to be robust to these crafted perturbations.

In this work, we extend the traditional PGD algorithm to operate within custom-defined hyper-rectangles $\mathbb{H}$ instead of the standard ε-ball subspace, which will be defined in the next section. The main modification lies in the definition of the step size: instead of using a scalar γ for a uniform step in all dimensions, we transform γ into a vector in $\mathbb{R}^m$, allowing the step size to vary across different dimensions. As a result, the perturbation step becomes an element-wise multiplication of the step size vector with the gradient's sign. This approach allows for more targeted adversarial training by accommodating variable sensitivity across different features of the input, which is especially useful in domains where certain dimensions of the input space are more susceptible to attack.

2.1.4 Formal Definitions of Robustness and Training Techniques

We now present key robustness properties that have been explored in computer vision and other domains, along with associated training techniques. These definitions are foundational to this thesis, and illustrate how design choices impact robustness outcomes.

Classification Robustness (CR). A common strategy to improve robustness is *data augmentation* [25], where one enriches the training set with perturbed samples that preserve the original label (e.g., by applying rotations or injecting noise into images). Given a neural network $N : \mathbb{R}^m \to \mathbb{R}^n$ and a training pair $(\hat{x}, y)$, we say the network is classifica-

tion robust if the predicted class remains unchanged under small perturbations:

Definition 2 (Classification Robustness)

$$CR(\varepsilon, \hat{x}) := \forall x : |x - \hat{x}|_p \leq \varepsilon \Rightarrow \arg\max N(x) = y$$

To train for CR, the following choices must be made:

c1. Perturbation radius ε,

c2. Distance metric $|\cdot - \cdot|_p$,

c3. Perturbation generation strategy (e.g., sampling, generative models, adversarial examples).

While intuitive, this formulation can be overly strict for samples near decision boundaries—where minor perturbations might justifiably change the class. For datasets that contain a significant number of such inputs, enforcing CR may lead to a significant reduction in accuracy.

Standard Robustness (SR). To account for such "uncertain" samples, *standard robustness* requires that the output does not vary by more than δ under perturbations. This is typically enforced through *adversarial training* [9]:

Definition 3 (Standard Robustness)

$$SR(\varepsilon, \delta, \hat{x}) := \forall x : |x - \hat{x}| \leq \varepsilon \Rightarrow |N(x) - N(\hat{x})| \leq \delta$$

The standard loss $\mathscr{L}(\hat{x}, y)$ is replaced with a worst-case objective:

$$\max_{x:|x-\hat{x}|\leq\varepsilon} \mathscr{L}(x, y)$$

This is commonly approximated using FGSM or PGD attacks (see Section 2.1.3). Design choices **c1**–**c3** still apply.

Lipschitz Robustness (LR). Another competing robustness definition is *Lipschitz robustness*, which is inspired by the well-established concept of Lipschitz continuity and bounds output variation relative to the input variation by a constant L [26]:

Definition 4 (Lipschitz Robustness)

$$LR(\varepsilon, L, \hat{x}) := \forall x : |x - \hat{x}| \leq \varepsilon \Rightarrow |N(x) - N(\hat{x})| \leq L|x - \hat{x}|$$

Training for LR can involve constrained optimisation via semi-definite programming [27], or including a projection step that restricts the weight matrices to those with suitable Lipschitz constants [28].

Strong Classification Robustness (SCR). Finally, constraint-based training methods have attempted to generalise robustness definitions via logical constraints [29, 30], encoded as differentiable loss functions $\mathscr{L}_C$. However, the use of non-differentiable operators like arg max limits their applicability to CR. Instead, surrogate formulations such as *strong classification robustness* are used:

Definition 5 (Strong Classification Robustness)

$$SCR(\varepsilon, \eta, \hat{x}) := \forall x : |x - \hat{x}| \leq \varepsilon \Rightarrow N_y(x) \geq \eta$$

Here, $N_y(x)$ is the score assigned to the true class, and η is a fixed threshold (e.g., 0.52 in [30]).

Each of these definitions can also be used as a loss function during training [30]. In practice, constraint loss functions are integrated into training as:

$$\mathscr{L}^*(\hat{x}, y) = \alpha \mathscr{L}(\hat{x}, y) + \beta \mathscr{L}_C(\hat{x}, y)$$

where $\mathscr{L}_C$ encodes the logical constraint, and α, β control its weight relative to standard loss. While this appears general, constraints involving the true label y rather than $N(\hat{x})$ risk entangling robustness and accuracy. Moreover, SCR avoids the use of arg max, which limits its expressiveness for enforcing strict CR.

These definitions are not equivalent: some are strictly stronger, others incomparable. Chapter 3 explores these relationships formally and experimentally. As will be shown, training for one property does not guarantee another—highlighting the need to align robustness objectives with verification strategies.

2.1.5 Geometric Analysis of Embedding Spaces

In recent years, the study of manifold subspaces has gained significant attention in the context of machine learning verification [31], as the geometric properties of data regions play a crucial role in analysing model robustness. Since we investigate applications from computer vision, NLP and NIDS, each involving different types of data (as discussed in Section 2.1.1), we define a subspace $\mathscr{S} \subset \mathbb{R}^m$ in relation to the input space $\mathbb{R}^m$ of a neural network $N : \mathbb{R}^m \to \mathbb{R}^n$. In this section, we formally define the most common subspaces used in verification: convex sets, convex hulls, zonotopes, and hyper-rectangles (also known as multi-dimensional intervals), following the definitions in [31].

Definition 6 (Convex Set) *A set $\mathbb{Z} \subseteq \mathbb{R}^m$ is said to be* convex *if, for any two points $x_1, x_2 \in \mathbb{Z}$, the line segment joining them is entirely contained within $\mathbb{Z}$. Formally, this means that for all $x_1, x_2 \in \mathbb{Z}$ and $\lambda \in [0,1]$, the points*

$$\lambda x_1 + (1-\lambda)x_2 \in \mathbb{Z}.$$

In other words, a set is convex if, for any pair of points in the set, the entire segment connecting them lies within the set.

The convex hull of a set is the smallest convex set that contains all the points of the set. It can be seen as the "tightest" boundary enclosing the points.

Definition 7 (Convex Hull) *The* convex hull *of a set $\mathbb{Z} \subseteq \mathbb{R}^m$, denoted by $conv(\mathbb{Z})$, is the smallest convex set containing $\mathbb{Z}$. Formally, it can be defined as:*

$$conv(\mathbb{Z}) := \left\{ \sum_{i=1}^{p} \lambda_i x_i \;\middle|\; x_i \in \mathbb{Z},\ \lambda_i \geq 0,\ \sum_{i=1}^{p} \lambda_i = 1,\ p \in \mathbb{N} \right\}.$$

In other words, $\text{conv}(\mathbb{Z})$ consists of all finite convex combinations of points in $\mathbb{Z}$. The construction of the convex hull has a complexity of $O(p^{m/2})$, where p is the number of points and m is the number of dimensions.

A zonotope is a geometric shape formed by the Minkowski sum of line segments.

Definition 8 (Zonotope) *Given a center $c \in \mathbb{R}^m$ and generators $g_1, \ldots, g_p$, a zonotope is*

$$\mathscr{Z} := \left\{ c + \sum_{i=1}^{p} \lambda_i g_i \;\middle|\; \lambda_i \in [-1,1], \forall i \in [1,\ldots,p] \right\}.$$

Zonotopes are computationally more efficient than convex hulls, with a construction complexity of $O(m \cdot p)$, where m is the dimensionality and p is the number of generators.

Finally, an interval is a simple shape defined by lower and upper bounds for each dimension, and it is equivalent to a multi-dimensional rectangle (or hyper-rectangle). Intervals are easy to construct with a complexity of $O(m)$, where m is the dimensionality, and are often used in verification.

Definition 9 (Interval (aka Hyper-Rectangle)) *Given a lower and upper bound $\underline{x}, \overline{x} \in \mathbb{R}^m$ such that $\underline{x}_{(i)} \leq \overline{x}_{(i)} \forall i \in 1,\ldots,m$, a multi-dimensional interval $\mathscr{I} \subset \mathbb{R}^m$ is*

$$\mathscr{I} := \left\{ x \in \mathbb{R}^m \;\middle|\; \underline{x}_{(i)} \leq x_{(i)} \leq \overline{x}_{(i)}, \forall i \in [1,\ldots,m] \right\}.$$

Table 2.1 summarises the construction complexities of these different shapes. Ideally, convex hulls would be the preferred choice due to their precise and detailed representations of subspaces. However, their computational complexity renders them infeasible in high dimensions. Zonotopes provide a promising alternative, as they are more precise than hyper-rectangles while remaining computationally tractable. Despite their theoretical compatibility with complete verifiers, practical limitations arise because most state-of-the-art verifiers do not support zonotopes. Hyper-rectangles, or intervals, are the simplest to construct and are supported by all verifiers, making them the default choice in many verification pipelines.

Shape	**Construction Complexity (Big-O)**
Convex Hull	$O(p^{m/2})$
Zonotope	$O(m \cdot p)$
Interval	$O(m)$

Table 2.1: *Construction complexity for different geometric shapes, where m is the number of dimensions and p is the number of points or generators.*

Visualising the Verifiability–Generalisability Trade-Off. In NLP verification, a central challenge lies in balancing two conflicting objectives: *verifiability*, the ability to formally prove robustness over a given subspace, and *generalisability*, the extent to which that subspace contains meaningful, semantically coherent inputs (see Section 4.3.1). This trade-off is particularly relevant when defining geometric subspaces in embedding space.

Figure 2.1 illustrates this issue in 2D embedding space. Small ε-balls (a) are easier to verify but often fail to capture unseen valid sentences. Convex hulls (b) more tightly capture semantic variation but are computationally expensive in high dimensions. Hyper-rectangles (c) and their rotated variants (d) offer a middle ground between tractability and semantic coverage. The red points represent training-set sentences from a given class; the turquoise ones are test-set sentences from the same class. A useful subspace should contain both, while remaining verifiable.

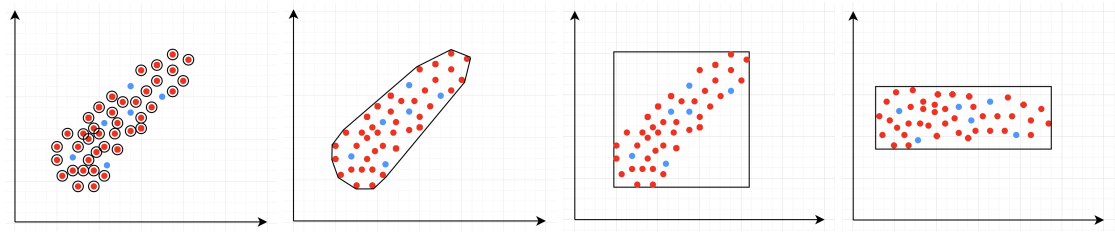


Figure 2.1: *(a) A verifiable but poorly generalisable ε-ball; (b) convex hull; (c) hyper-rectangle; (d) rotated hyper-rectangle. Red = training embeddings; turquoise = test embeddings of the same class.*

2.1.6 Specification Languages for Neural Network Verification

A central component of neural network verification is the ability to formally express the properties that models are expected to satisfy. In this section, we review two specification formats: `VNNLib`, widely used in formal verification competitions and supported by many tools, and `Vehicle`, a high-level language for compositional property definitions.

VNNLib. The VNNLib format is the standard for representing specifications in neural network verification competitions (VNN-COMP [32]) and is supported by most existing verifiers including Marabou [33], ERAN [34], and $\alpha\beta$-CROWN [35, 36]. It is a low-level language based on SMT-LIB [37] syntax, designed to encode input bounds and output constraints using basic arithmetic logic.

To illustrate its usage, we show in Figure 2.2 an encoding of Property 3 [38] from the

ACASXu [39] benchmark suite. The specification asserts that when an intruder aircraft is directly ahead and approaching, the "Clear-of-Conflict" (COC) manoeuvre must not be advised.

```
; ACASXu Property 3: COC should not be chosen when
; intruder is directly ahead and moving toward ownship

(declare-const X_0 Real) ; distance
(declare-const X_1 Real) ; angle
(declare-const X_2 Real) ; intruder heading
(declare-const X_3 Real) ; ownship speed
(declare-const X_4 Real) ; intruder speed

(declare-const Y_0 Real) ; Clear Of Conflict score
(declare-const Y_1 Real) ; Weak Left
(declare-const Y_2 Real) ; Weak Right
(declare-const Y_3 Real) ; Strong Left
(declare-const Y_4 Real) ; Strong Right

; input conditions
(assert (>= X_0 1500.0))
(assert (<= X_0 1800.0))
(assert (>= X_1 -0.06))
(assert (<= X_1 0.06))
(assert (>= X_2 3.10))
(assert (>= X_3 980.0))
(assert (>= X_4 960.0))

; COC not minimal
(assert (<= (- Y_0 Y_1) 0.0))
(assert (<= (- Y_0 Y_2) 0.0))
(assert (<= (- Y_0 Y_3) 0.0))
(assert (<= (- Y_0 Y_4) 0.0))
```

Figure 2.2: *A VNNLib specification of ACASXu Property 3: COC must not be chosen when the intruder is directly ahead and moving toward the ownship.*

Vehicle. Vehicle [40, 41, 42] is a system for embedding logical specifications into neural networks. At its heart is the Vehicle specification language, a high-level, functional language for writing mathematically-precise specifications for your networks, which can be compiled down to low-level neural network verifiers such as Marabou.

Key components of Vehicle include:

- `@network`: Declares the neural network model under verification, including its input/output types.
- `@dataset`: Enables referencing external datasets directly within the specification, allowing scalable property definitions over real inputs without manual replication.
- `@parameter`: Declares symbolic constants whose values can be supplied at compile time, supporting configurable and reusable specifications.
- `@property`: Defines the logical property or constraint to be verified; each `@property` represents a top-level assertion to be checked by the backend verifier.

Figure 2.3 shows how Property 3 from ACASXu can be expressed in Vehicle. Compared to the VNNLib version, this specification introduces symbolic names and reusable predicates, improving clarity and extensibility. It also allows users to declare normalisation functions directly within the specification, which in some cases helps address the embedding gap by mapping inputs to a canonical scale before verification. Note how lines 29–39 introduce a series of logical constraints on the input space — these correspond to global properties in the sense of Chapter 5, where rule-based subspaces are encoded similarly for verification and training alignment.

```
1  -- ACASXu Property 3: COC should not be chosen when
2  -- intruder is directly ahead and moving toward ownship
3
4  -- Define a name for the type of inputs and outputs of the network
5  type InputVector = Vector Rat 5
6  type OutputVector = Vector Rat 5
7
8  -- Add meaningful names for the indices.
9  distanceToIntruder = 0   -- measured in metres
10 angleToIntruder    = 1   -- measured in radians
11 intruderHeading    = 2   -- measured in radians
12 speed              = 3   -- measured in metres/second
13 intruderSpeed      = 4   -- measured in meters/second
14
15 clearOfConflict = 0
16 weakLeft        = 1
17 weakRight       = 2
18 strongLeft      = 3
19 strongRight     = 4
20
21 -- Define the network.
22 @network
23 acasXu : InputVector -> OutputVector
24
25 -- Define a constraint that says the network chooses output `i' when given the input `x'
26 advises : Index 5 -> InputVector -> Bool
27 advises i x = forall j . i != j => acasXu x ! i < acasXu x ! j
28
29 -- Define the `directly ahead' and `moving towards' predicates
30 directlyAhead : InputVector -> Bool
31 directlyAhead x =
32   1500  <= x ! distanceToIntruder <= 1800 and
33   -0.06 <= x ! angleToIntruder    <= 0.06
34
35 movingTowards : InputVector -> Bool
36 movingTowards x =
37   x ! intruderHeading >= 3.10  and
38   x ! speed           >= 980   and
39   x ! intruderSpeed   >= 960
40
41 -- Assert that the score for COC will not be minimal
42 @property
43 property3 : Bool
44 property3 = forall x . validInput x and directlyAhead x and movingTowards x =>
45   not (advises clearOfConflict x)
```

Figure 2.3: *A Vehicle specification for ACASXu Property 3: under approach conditions, the network must not advise COC as the minimal output.*

Use of Specification Languages in this Thesis. As discussed in the previous section, this work primarily relies on axis-aligned hyper-rectangles to define input subspaces.

These geometric regions are straightforward to encode using formats like VNNLib, which are supported by most mainstream verifiers. As a result, we do not require a high-level specification language for the properties used in the computer vision and NLP domains.

However, there are cases where hyper-rectangles are inadequate or impractical:

- Some properties cannot be expressed as axis-aligned boxes at all. For example, the safety properties used in the Alsomitra drone controller [43] involve constraints where bounds on one feature depend on the values of others [10]. Such relational constraints define geometric regions like half-spaces, which cannot be encoded as hyper-rectangles that require fixed upper and lower bounds per dimension.

- Other properties could, in principle, be represented as unions of multiple hyper-rectangles, but the size of the specification required would be prohibitively large. This occurs in our NIDS case study (Chapter 5), where several specifications would require an unmanageable number of disjoint subregions to cover the intended property space.

To address these limitations, we use the Vehicle specification language in Chapter 5. Vehicle allows us to symbolically express such complex properties without enumerating explicit rectangular regions, enabling scalable and maintainable verification workflows.

While not all experiments in this thesis use Vehicle as the back-end (e.g., some rely directly on lower-level verifiers such as Marabou [33], ERAN [34], and Alpha-beta-CROWN [35]), we adopt Vehicle consistently for specification examples across the thesis. Its expressive and verifier-agnostic syntax provides a unified way to present both robustness and global properties across domains. In particular, in Chapters 3 and 4, we provide illustrative Vehicle specifications (Figures 3.5 and 4.3) to show how analogous properties could be encoded using a higher-level specification language.

2.1.7 Local vs Global Properties

A key distinction in this thesis is between *local* and *global* properties, which differ in how they are defined over subspaces of the input domain [44, 45].

Local properties refer to guarantees around a specific input point. Formally, given a neural network $N : \mathbb{R}^m \to \mathbb{R}^n$ and an input $\hat{x} \in \mathbb{R}^m$, a local property specifies the network's behaviour on a norm ball $\mathbb{B}(\hat{x}, \varepsilon) := \{x \in \mathbb{R}^m : |\hat{x} - x| \leq \varepsilon\}$ of uniform radius ε around

$\hat{x}$ (when the norm is ℓ_∞, the ball coincides with an axis-aligned ε-cube; for other norms, balls and boxes differ). This is the dominant setting in adversarial robustness [9, 46] and has been widely studied in formal verification [45, 34].

Global properties refer to guarantees over axis-aligned hyper-rectangles defined by per-dimension intervals (see Definition 9). Global properties are important in safety-critical systems [47, 48, 49], and are often encoded using symbolic constraints or logic-based specifications. An example comes from the ACAS Xu collision avoidance system [39, 38], where global properties require that the network avoid selecting unsafe actions in entire regions of the state space, such as when an intruder aircraft is directly ahead and closing in. These types of constraints can span large portions of the input domain.

We treat both cases under a region-centric formulation: local properties operate on norm balls, while global properties operate on hyper-rectangles. These shapes are distinct in general (except with ℓ_∞ norm), but this viewpoint allows a single specification-first training–verification methodology to be applied across both settings.

Chapter 3 studies local robustness notions in vision. Chapter 4 uses both local norm balls (baselines) and global hyper-rectangles constructed from paraphrase sets in NLP. Chapter 5 focuses on global, rule-based hyper-rectangles in NIDS.

2.2 Literature Review

In this section, we review the literature related to this thesis. We will first focus on robust training and DNN verification, followed by NLP robustness and verification, NIDS-related work, and ending with a systematic view of DNN robustness and verification pipelines.

2.2.1 Robust Training

Verifying DNNs poses significant challenges if they are not appropriately trained. The fundamental issue, as previously mentioned, lies in the failure of DNNs, including even sophisticated models, to meet essential verification properties, such as *robustness* [50]. To enhance robustness, various training methodologies have been proposed. It is noteworthy that, although robust training by *projected gradient descent* [1, 9, 51] predates

verification, contemporary approaches are often related to, or derived from, the corresponding verification methods by optimising verification-inspired regularisation terms or injecting specific data augmentation during training. In practice, after robust training, the model usually achieves higher certified robustness and is more likely to satisfy the desired verification properties [50]. Thus, robust training is a strong complement to robustness verification approaches.

Robust training techniques can be classified into several large groups:

- data augmentation [52],
- adversarial training [1, 9] including property-driven training [30, 53],
- IBP training [54, 55] and other forms of certified training [56], or
- a combination thereof [57, 50].

Data augmentation involves the creation of synthetic examples through the application of diverse transformations or perturbations to the initial training data. These generated instances are then incorporated into the original dataset to enhance the training process. *Adversarial training* entails identifying worst-case examples at each epoch during the training phase and calculating an additional loss on these instances. State of the art adversarial training involves projected gradient descent algorithms such as FGSM [1] and PGD [9]. *Certified training* methods focus on providing mathematical guarantees about the model's behaviour within certain bounds. Among them, we can name IBP training [54, 55] techniques, which impose intervals or bounds on the predictions or activations of the model, ensuring that the model's output lies within a specific range with high confidence.

Note that all techniques mentioned above can be categorised based on whether they primarily *augment the data* (such as data augmentation) or *augment the loss function* (as seen in adversarial, IBP and certified training). Augmenting the data tends to enhance generalisation and is efficient, although it may not help against stronger adversarial attacks. Conversely, methods that manipulate the loss functions directly are more resistant to strong adversarial attacks but often come with higher computational costs. Ultimately, the choice between altering data or loss functions depends on the specific requirements of the application and the desired trade-offs between performance, computational complexity, and robustness guarantees.

2.2.2 DNN Verification

Formal verification is an active field across several domains including hardware [58, 59], software languages [60], network protocols [61] and many more [62]. However, it was only recently that this became applicable to the field of machine learning [38]. An input query to a verifier consists of a subspace within the input space and a target subspace of outputs, typically a target output class. The verifier then returns either *true*, *false* or *unknown*. *True* indicates that there exists an input within the given input subspace whose output falls within the given output subspace, often accompanied by an example of such input. *False* indicates that no such input exists. Several verifiers are popular in DNN verification and competitions [63, 64, 65, 66]. We can divide them into 2 main categories: complete verifiers which return *true/false* and incomplete verifiers which return *true/unknown*. While complete verifiers are always deterministic, incomplete verifiers may be probabilistic. Unlike deterministic verification, probabilistic verification is not sound and a verifier may incorrectly output *true* with a very low probability (typically 0.01%).

Complete Verification based on Linear Programming & Satisfiability Modulo Theories (SMT) Solving. Generally, SMT solving is a group of methods for determining the satisfiability of logical formulas with respect to underlying mathematical theories such as real arithmetic, bit-vectors, or arrays [67]. These methods extend traditional satisfiability (SAT) solving by incorporating domain-specific reasoning, making them particularly useful for verifying complex systems. In the context of neural network verification, SMT solvers encode network behaviours and safety properties as logical constraints, enabling rigorous checks for violations of specifications [68]. When the activation functions are piecewise linear (e.g. ReLU), the DNN can be encoded by conjunctions and disjunctions of linear inequalities and thus linear programming algorithms can be directly applied to solve the satisfiability problem. A state-of-the-art tool is Marabou [33], which answers queries about neural networks and their properties in the form of constraint satisfaction problems. Marabou takes the network as input and first applies multiple pre-processing steps to infer bounds for each node in the network. It applies the algorithm ReLUplex [38], a combination of *Simplex* [69] search over linear constraints, modified to work for networks with piece-wise linear activation functions. With time, Marabou grew into a complex prover with multiple heuristics supplementing the original ReLUplex algorithm [33], for example it now includes mixed-integer linear programming

(MILP) [70] and abstract interpretation based algorithms which we survey below. MILP-based approaches [71, 72, 73] encode the verification problem as a mixed-integer linear programming problem, in which the constraints are linear inequalities and the objective is represented by a linear function. Thus, the DNN verification problem can be precisely encoded as a MILP problem. For example, ERAN [74] combines abstract interpretation with the MILP solver GUROBI [75]. By the time Branch and Bound (BaB) methodologies are introduced later, it becomes evident that the verification community has effectively consolidated diverse approaches into a unified taxonomy. Modern verifiers, such as $\alpha\beta$-CROWN [35, 36], take full advantage of this combination and effectively balance efficiency with precision.

Incomplete Verification based on Abstract Interpretation takes inspiration from the domain of abstract interpretation, and mainly uses linear relaxations on ReLU neurons, resulting in an over-approximation of the initial constraint. Abstract interpretation was first developed by Cousot and Cousot [76] in 1977. It formalises the idea of abstraction of mathematical structures, in particular those involved in the specification of properties and proof methods of computer systems [77] and it has since been used in many applications [3]. Specifically, for DNN verification, this technique can model the behaviour of a network using an abstract domain that captures the possible range of values the network can output for a given input. Abstract interpretation-based verifiers can define a lower bound and an upper bound of the output of each ReLU neuron as linear constraints, which define a region called ReLU polytope that gets propagated through the network. To propagate the bounds, one can use *interval bound propagation* (IBP) [46, 54, 78, 79]. The strength of IBP-based methods lies in their efficiency; they are faster than alternative approaches and demonstrate superior scalability. However, their primary limitation lies in the inherently loose bounds they produce [54]. This drawback becomes particularly pronounced in the case of deeper neural networks, typically those with more than 10 layers [80], where they cannot certify non-trivial robustness properties. Other methods that are less efficient but produce tighter bounds are based on polyhedra abstraction, such as CROWN [81] and DeepPoly [34], or based on multi-neuron relaxation, such as PRIMA [82]. An abstract interpretation tool CORA [31], uses polyhedral abstractions and reachability analysis for formal verification of neural networks. It integrates various set representations, such as zonotopes, and algorithms to compute reachable sets for both continuous and hybrid systems, providing tighter bounds in verification tasks. Another

mature tool in this category is ERAN [74], which uses abstract domains (DeepPoly) with custom multi-neuron relaxations (PRIMA) to support fully-connected, convolutional, and residual networks with ReLU, Sigmoid, Tanh, and Maxpool activations. Note that, having lost completeness, they can work with a more general class of neural networks (e.g. neural networks with non linear layers).

Modern Neural Network Verifiers. Modern verifiers are complex tools that take advantage of a combination of complete and incomplete methods as well as additional heuristics. The term Branch and Bound (BaB) [47, 83, 84, 85, 86, 36, 87] often refers to the method that relies on the piecewise linear property of DNNs: since each ReLU neuron outputs $\text{ReLU}(x) = \max\{x,0\}$ is piecewise linear, we can consider its two linear pieces $x \geq 0$, $x \leq 0$ separately. A BaB verification approach, as the name suggests, consists of two parts: branching and bounding. It first derives a lower bound and an upper bound, then, if the lower bound is positive it terminates with 'verified', else, if the upper bound is non-positive it terminates with 'not verified' (*bounding*). Otherwise, the approach recursively chooses a neuron to split into two branches (*branching*), resulting in two linear constraints. Then bounding is applied to both constraints and if both are satisfied the verification terminates, otherwise the other neurons are split recursively. When all neurons are split, the branch will contain only linear constraints, and thus the approach applies linear programming to compute the constraint and verify the branch. It is important to note that BaB approaches themselves are neither inherently complete nor incomplete. BaB is an algorithm for splitting problems into sub-problems and requires a solver to resolve the linear constraints. The completeness of the verification depends on the combination of BaB and the solver used. *Multi-Neuron Guided Branch-and-Bound (MN-BaB)* [85] is a state-of-the-art neural network verifier that builds on the tight multi-neuron constraints proposed in PRIMA [88] and leverages these constraints within a BaB framework to yield an efficient, GPU based dual solver. Another state-of-the-art tool is $\alpha\beta$-CROWN [35, 36], a neural network verifier based on an efficient linear bound propagation framework and branch-and-bound. It can be accelerated efficiently on GPUs and can scale to relatively large convolutional networks (e.g., 10^7 parameters). It also supports a wide range of neural network architectures (e.g., CNN, ResNet, and various activation functions).

Probabilistic incomplete verification approaches add random noise to models to smooth them, and then derive certified robustness for these smoothed models. This field is com-

monly referred to as Randomised Smoothing, given that these approaches provide probabilistic guarantees of robustness, and all current probabilistic verification techniques are tailored for smoothed models [89, 90, 91, 92, 93, 94]. Given that this work focuses on deterministic approaches, here we only report the existence of this line of work without going into details.

Note that these existing verification approaches primarily focus on computer vision tasks, where images are seen as vectors in a continuous space and every point in the space corresponds to a valid image, while sentences in NLP form a discrete domain, making it challenging to apply traditional verification techniques effectively.

In this thesis we use both an abstract interpretation-based incomplete verifier (ERAN [74]) and an SMT-based complete verifier (Marabou [33]) in order to demonstrate the effect that the choice of a verifier may bring, and demonstrate common trends.

2.2.3 Robustness and Verification in Computer Vision

Robustness is a longstanding and extensively studied property in computer vision, particularly in the context of deep learning. Since the discovery of adversarial examples [2, 1], ensuring that image classifiers behave reliably under input perturbations has become a central challenge in both training and verification.

Initial approaches focused on training methods to improve robustness against adversarial attacks, such as adversarial training [9], defensive distillation [95], and regularisation-based techniques [96]. These methods aimed to improve robustness against perturbations bounded by some norm (e.g., ℓ_∞, ℓ_2), typically evaluated empirically via attack success rates. However, as attacks improved [97, 98], it became clear that empirical robustness was insufficient: formal guarantees were required.

In response, verification-based techniques emerged to provide formal guarantees. Notable approaches include abstract interpretation [47], SMT/LP-based methods [38], and bounding techniques such as interval bound propagation (IBP) [99]. These methods aim to prove, for a given input and perturbation region, that the classifier's output remains stable. State-of-the-art verifiers such as Alpha-beta-CROWN [35], Marabou [33], and others evaluated in recent robustness benchmarks [32] continue to push the frontier of scalability and precision.

Despite this progress, robustness verification remains an open challenge. Three major challenges persist:

- **Geometric**: The input space of images is high-dimensional and continuous, but the decision boundaries of neural networks can be highly non-linear and fractured. Understanding how perturbations interact with these boundaries is non-trivial.
- **Training/Optimisation**: Training for provable robustness often reduces accuracy [100, 101]. Balancing accuracy and robustness remains a key trade-off.
- **Logical**: Different definitions of robustness (e.g., SR, LR, CR, SCR) lead to different formal properties, yet are often treated interchangeably. This can cause mismatches between training, testing, and verification goals.

This thesis addresses the third challenge. In Chapter 3, we present a formal and empirical analysis of several commonly used robustness definitions in image classification. Whereas previous work often focuses on a single property or treats them as equivalent, we provide a systematic comparison of their logical relationships — such as implication, containment, and incomparability — and demonstrate how mismatches between training and verification properties can lead to misleading robustness claims.

Our contribution complements the existing verification literature by clarifying how robustness should be defined and evaluated — particularly in simple domains like image classification, where embedding complexity is minimal. This conceptual foundation sets the stage for tackling more complex domains such as NLP and NIDS in subsequent chapters.

2.2.4 NLP Robustness

There exists a substantial body of research dedicated to enhancing the adversarial robustness of NLP systems [102, 103, 104, 105, 106, 107, 108]. These efforts aim to mitigate the vulnerability of NLP models to adversarial attacks and improve their resilience in real-world scenarios [103, 104] and mostly employ data augmentation techniques [109, 110]. In NLP, we can distinguish perturbations based on three main criteria:

- where and how the perturbations occur,
- whether they are altered automatically using some defined rules (vs. generated by humans or LLMs) and
- whether they are adversarial (as opposed to random).

In particular, perturbations can occur at the character, word, or sentence level [111, 112, 113] and may involve deletion, insertion, swapping, flipping, substitution with synonyms, concatenation with characters or words, or insertion of numeric or alphanumeric characters [114, 115, 116]. For instance, in character level adversarial attacks, Belinkov et al. [117] introduce natural and synthetic noise to input data, while Gao et al. [118] and Li et al. [119] identify crucial words within a sentence and perturb them accordingly. For word level attacks, they can be categorised into gradient-based [114, 120], importance-based [121, 122], and replacement-based [123, 124, 125] strategies, based on the perturbation method employed. Moreover, Moradi et al. [126] introduce rule-based non-adversarial perturbations at both the character and word levels. Their method simulates various types of noise typically caused by spelling mistakes, typos, and other similar errors. In sentence level adversarial attacks, some perturbations [127, 128] are created so that they do not impact the original label of the input and can be incorporated as a concatenation in the original text. In such scenarios, the expected behaviour from the model is to maintain the original output, and the attack can be deemed successful if the label/output of the model is altered. Additionally, non-rule-based sentence perturbations can be obtained through prompting LLMs [129, 130] to generate rephrasing of the inputs. By augmenting the training data with these perturbed examples, models are exposed to a more diverse range of linguistic variations and potential adversarial inputs. This helps the models to generalise better and become more robust to different types of adversarial attacks. To help with this task, the NLP community has gathered a dataset of adversarial attacks named AdvGLUE [131], which aims to be a principled and comprehensive benchmark for NLP robustness measurements.

In this work we employ a PGD-based adversarial training as the method to enhance the robustness and verifiability of our models against gradient-based adversarial attacks. For non-adversarial perturbations, we create rule-based perturbations at the character and word level as in Moradi et al. [126] and non-rule-based perturbations at the sentence level using PolyJuice [129] and Vicuna [130]. We thus cover most combinations of the three choices above (bypassing only human-generated adversarial attacks as there is no sufficient data to admit systematic evaluation).

Method	Datasets	NLP perturbations	Embeddings	Architectures (# of parameters)	Robust training	Verification algorithm	Verification characteristics
Ours	RUARobot, Medical	General purpose: char, word and sentence perturbations, ϵ-ball	Sentence: S-BERT, S-GPT	FFNN (10^4)	**PGD**-based loss function augmentation	**SMT**, **BaB**, Abstract interpretation	**Complete**, **Deterministic**
Jia et al. (2019) [132]	IMDB, SNLI	Word substitution	Word: GloVe	LSTM, CNN, BoW, Attention-based, (10^5)	**IBP**-based loss function augmentation	Abstract Interpretation IBP	Incomplete, **Deterministic**
Huang et al. (2019) [133]	AGNews, SST	Char and word substitution	Word: GloVe	CNN (10^5)	**IBP**-based loss function augmentation	Abstract Interpretation IBP	Incomplete, **Deterministic**
Welbl et al. (2020) [134]	SNLI, MNLI	Word deletion	Word: GloVe	Attention-based (10^5)	Data augmentation, random and beam search adversarial training, **IBP**-based	Abstract Interpretation IBP	Incomplete, **Deterministic**
Zhang et al. (2021) [135]	IMDB, SST, SST2	Word perturbations	Word: not specified	LSTM (10^5)	**IBP**-based loss function augmentation	Abstract Interpretation IBP	Incomplete, **Deterministic**
Wang et al. (2023) [136]	IMDB, YELP, SST2	Word substitution	Word: GloVe	CNN (10^5)	**IBP**-based: Embedding Interval Bound Constraint (EIBC) triplet loss	Abstract Interpretation IBP	Incomplete, **Deterministic**
Ko et al. (2019) [137]	CogComp QC	ϵ-ball	Word: not specified	RNN, LSTM (10^5)	-	Abstract Interpretation IBP	Incomplete, **Deterministic**
Shi et al. (2020) [138]	YELP, SST	ϵ-ball	Word: not specified	Transformer (10^6)	-	Abstract Interpretation IBP	Incomplete, **Deterministic**
Du et al. (2021) [139]	Rotten Tomatoes Movie Review, Toxic Comment	ϵ-ball	Word: GloVe	RNN, LSTM (10^5)	**Zonotope**-based loss function augmentation	Abstract Interpretation Zonotopes	Incomplete, **Deterministic**
Bonaert et al. (2021) [140]	SST, YELP	ϵ-ball	Word: not specified	Transformer (10^6)	-	Abstract Interpretation Zonotopes	Incomplete, **Deterministic**

Table 2.2: *Summary of the main features of the existing NLP verification approaches. In bold are state-of-the-art methods.*

Method	Datasets	NLP perturbations	Embeddings	Architectures (# of parameters)	Robust training	Verification algorithm	Verification characteristics
Ye et al. (2020) [141]	IMDB, Amazon	Word substitution	Word: GloVe	Transformer (10^8)	Data augmentation	Randomised smoothing ($\alpha = 0.01, n = 5000$)	Incomplete, Probabilistic
Wang et al. (2021) [142]	IMDB, AGNews	Word substitution	Word: GloVe	LSTM (10^5)	Data augmentation	Differential privacy	Incomplete, Probabilistic
Zhao et al. (2022) [143]	AGNews, SST	Word substitution	Word: GloVe	Transformer (10^8)	Data augmentation and IBP-based	Randomised smoothing ($\alpha = 0.001, n = 30050$)	Incomplete, Probabilistic
Zeng et al. (2023) [144]	IMDB, YELP	Char and word substitution	Word: not specified	Transformer (10^8)	Data augmentation	Randomised smoothing ($\alpha = 0.05, n = 5000$)	Incomplete, Probabilistic
Ye et al. (2023) [145]	IMDB, SST2, YELP, AGNews	Word substitution	Word: not specified	Transformer (10^8)	Data augmentation	Randomised smoothing ($\alpha = 0.001, n = 9000$)	Incomplete, Probabilistic
Zhang et al. (2023) [146]	IMDB, Amazon, AGNews	Word perturbations	Word: GloVe	LSTM, Transformer (10^8)	Data augmentation	Randomised smoothing ($\alpha = 0.001, n = 20000$)	Incomplete, Probabilistic
Zhang et al. (2023) [147]	SST2, AGNews	Word perturbations	Word: not specified	Transformer (10^9)	-	Randomised smoothing ($\alpha = 0.05, n = 5000$)	Incomplete, Probabilistic

Table 2.3: *Summary of the main features of the existing randomised smoothing approaches.*

2.2.5 Previous NLP Verification Approaches

Although DNN verification studies have predominantly focused on computer vision, there is a growing body of research exploring the verification of NLP. This research can be categorised into three main approaches: using IBP, zonotopes, and randomised smoothing. Tables 2.2 and 2.3 show a comparison of these approaches. To the best of our knowledge, this work is the first to use an SMT-based verifier for this purpose, and compare it with an abstract interpretation-based verifier on the same benchmarks.

NLP Verification via Interval Bound Propagation. The first technique successfully adopted from the computer vision domain for verifying NLP models was the IBP. IBP was used for both training and verification with the aim to minimise the upper bound on the maximum difference between the classification boundary and the input perturbation region. It was achieved by augmenting the loss function with a term that penalises large perturbations. Specifically, IBP incorporates interval bounds during the forward propagation phase, adding a regularisation term to the loss function that minimises the distance between the perturbed and unperturbed outputs. This facilitated the minimisation of the perturbation region in the last layer, ensuring it remained on one side of the classification boundary. As a result, the adversarial region becomes tighter and can be considered certifiably robust. Notably, Jia et al. [132] proposed certified robust models on word substitutions in text classification. The authors employed IBP to optimise the upper bound over perturbations, providing an upper bound over the discrete set of perturbations in the word vector space. Similarly, POPQORN[137] introduced robustness guarantees for RNN-based networks by handling the non-linear activation functions of complicated RNN structures (like LSTMs and GRUs) using linear bounds. Later, Shi et al.[138] developed a verification algorithm for transformers with self-attention layers. This algorithm provides a lower bound to ensure the probability of the correct label remains consistently higher than that of the incorrect labels. Furthermore, Huang et al. [133] introduced a verification and verifiable training method with a tighter over-approximation in style of the Simplex algorithm [38]. To make the network verifiable, they defined the convex hull of all the original unperturbed inputs as a space of perturbations. By employing the IBP algorithm, they generated robustness bounds for each neural network layer. Later on, Welbl et al. [134] differentiated from the previous approaches by using IBP to address the under-sensitivity issue. They designed and formally verified the 'under-sensitivity specification' that a model should not become more confident as arbitrary subsets of input

words are deleted. Recently, Zhang et al. [135] introduced Abstract Recursive Certification (ARC) to verify the robustness of LSTMs. ARC defines a set of programmatically perturbed string transformations to construct a perturbation space. By memorising the hidden states of strings in the perturbation space that share a common prefix, ARC can efficiently calculate an upper bound while avoiding redundant hidden state computations. Finally, Wang et al. [136] improved on the work of Jia et al. by introducing Embedding Interval Bound Constraint (EIBC). EIBC is a new loss that constraints the word embeddings in order to tighten the IBP bounds.

The strength of IBP-based methods is their efficiency and speed, while their main limitation is the bounds' looseness, further accentuated if the neural network is deep.

NLP Verification via Propagating Zonotopes. Another popular verification technique applied to various NLP models is based on propagating zonotopes, which produces tighter bounds then IBP methods. One notable contribution in this area is Cert-RNN [139], a robust certification framework for RNNs that overcomes the limitations of POPQORN. The framework maintains inter-variable correlation and accelerates the non-linearities of RNNs for practical uses. Cert-RNN utilised zonotopes [148] to encapsulate input perturbations and can verify the properties of the output zonotopes to determine certifiable robustness. This results in improved precision and tighter bounds, leading to a significant speedup compared to POPQORN. Analogously, Bonaert et al. [140] propose DeepT, a certification method for large transformers. It is specifically designed to verify the robustness of transformers against synonym replacement-based attacks. DeepT employs multi-norm zonotopes to achieve larger robustness radii in the certification and can work with networks much larger than Shi et al.

Methods that propagate zonotopes produce much tighter bounds than IBP-based methods, which can be used with deeper networks. However, they use geometric methods and do not take into account semantic considerations (e.g. do not use semantic perturbations).

NLP Verification via Randomised Smoothing. Randomised smoothing [48] is another technique for verifying the robustness of deep language models that has recently grown in popularity due to its scalability [141, 142, 143, 144, 145, 146, 147]. The idea is to leverage randomness during inference to create a smoothed classifier that is more robust to small perturbations in the input. This technique can also be used to give certified guarantees against adversarial perturbations within a certain radius. Generally, randomised smoothing begins by training a regular neural network on a given dataset. During the

inference phase, to classify a new sample, noise is randomly sampled from the predetermined distribution multiple times. These instances of noise are then injected into the input, resulting in noisy samples. Subsequently, the base classifier generates predictions for each of these noisy samples. The final prediction is determined by the class with the highest frequency of predictions, thereby shaping the smoothed classifier. To certify the robustness of the smoothed classifier against adversarial perturbations within a specific radius centered around the input, randomised smoothing calculates the likelihood of agreement between the base classifier and the smoothed classifier when noise is introduced to the input. If this likelihood exceeds a certain threshold, it indicates the certified robustness of the smoothed classifier within the radius around the input.

The main advantage of randomised smoothing-based methods is their scalability, indeed recent approaches are tested on larger transformer such as BERT and Alpaca. However, their main issue is that they are probabilistic approaches, meaning they give certifications up to a certain probability. In this work we focus on deterministic approaches, hence we only report these works in Table 2.3 for completeness without delving deeper into each paper here. All randomised smoothing-based approaches use data augmentation obtained by semantic perturbations.

2.2.6 Datasets Used in NLP Verification

In this section, we first summarise the datasets that have been commonly used in the NLP verification literature. We then highlight the gaps these datasets present with respect to safety-critical applications, and describe the two NLP datasets used in this thesis.

Existing NLP Verification Datasets. Table 2.4 summarises the main features and tasks of the datasets used in NLP verification. Despite their diverse origins and applications, the datasets in the literature are usually binary or multi-class text classification problems. Furthermore, datasets can be sensitive to perturbations, i.e. perturbations can have non-trivial impact on label consistency. For example, Jia et al. [132] use IBP with the SNLI [149][1] dataset (see Tables 2.2 and 2.4) to show that word perturbations (e.g. 'good' to 'best') can change whether one sentence entails another. Some works such as Jia et al. [132] try to address this label consistency, while others do not.

Additionally, we find that the previous research on NLP verification does not utilise

[1] A semantic inference dataset that labels whether one sentence entails, contradicts or is neutral to another sentence.

safety critical datasets (which strongly motivates the choice of datasets in alternative verification domains), with the exception of Du et al. [139] that use the Toxic Comment dataset [150]. Other papers do not provide detailed motivation as to why the dataset choices were made, however it could be due to the datasets being commonly used in NLP benchmarks (IMDB etc.).

Dataset	Safety Critical	Category	Tasks	Size	Classes
IMDB [151]	×	Sentiment analysis	Document-level and sentence-level classification	25,000	2
SST [152]	×	Sentiment analysis	Sentiment classification, hierarchical sentiment classification, sentiment span detection	70,042	5
SST2 [152]	×	Sentiment analysis	Sentiment classification	70,042	2
YELP [153]	×	Sentiment analysis	Sentiment classification	570,771	2
Rotten Tomatoes Movie Review [154]	×	Sentiment analysis	Sentiment classification	48,869	3/4
Amazon [155]	×	Sentiment analysis	Sentiment classification, aspect-based sentiment analysis	34,686,770	5
SNLI [149]	×	Semantic inference	Natural language inference, semantic similarity	570,152	3
MNLI [156]	×	Semantic inference	Natural language inference, semantic similarity, generalisation	432,702	3
AGNews [157]	×	Text analysis	Text classification, sentiment classification	127,600	4
CogComp QC [158]	×	Text analysis	Question classification, semantic understanding	15,000	6/50
Toxic Comment [150]	✓	Text analysis	Toxic comment classification, fine-grained toxicity analysis, bias analysis	18,560	6

Table 2.4: *Summary of the main features of the datasets used in NLP verification.*

Datasets Used in This Thesis. In contrast to existing work, this thesis focuses on verification for safety-critical NLP settings, where failure may have regulatory, legal, or ethical consequences. We use two datasets that reflect real-world regulatory concerns:

- **R-U-A-Robot** [19] consists of 6,800 utterances prompting identity disclosure (e.g., "Are you a robot?"), divided in three classes: positive, negative and ambiguous (where it is unclear whether the system is required to state its identity). We collapse ambiguous examples into the positive class, following a precautionary principle: if identity is unclear, disclosure is still expected.

- **Medical Safety** [20]: 2,917 Reddit questions annotated for medical relevance and risk level. Risk ratings follow the World Economic Forum taxonomy, but we merge all medical risk levels into a single class. Labels are provided by expert annotators.

These datasets were chosen to reflect realistic safety concerns—chatbot disclosure mandates [159, 160, 161] and responsible handling of medical queries [162, 163]. The two datasets differ in semantic structure: R-U-A-Robot includes lexically varied but semantically similar prompts, while Medical Safety includes semantically diverse queries. These datasets are used and analysed in Chapter 4, where we study robustness verification in NLP applications.

2.2.7 NIDS Robustness and Verification

Network intrusion detection systems (NIDS) are cybersecurity tools designed to monitor and analyse network traffic to detect unauthorised access, misuse, or anomalies. Traditionally, NIDS have relied on signature-based detection, which matches traffic patterns against known attack signatures. However, this approach struggles with detecting novel or evolving threats. In response, researchers have increasingly turned to machine learning models, particularly DNNs, due to their ability to generalise from data and identify previously unseen attack patterns [164, 165, 166].

Despite their promise, DNN-based NIDS face challenges regarding robustness, especially in adversarial settings where small, carefully crafted changes to network traffic can fool the model. While significant research has focused on verifying the robustness of DNNs, particularly in the computer vision domain [38, 97], these methods often do not translate directly to NIDS, due to the structured and heterogeneous nature of network data, which includes both continuous and discrete features.

In this section, we review the literature on the robustness and verification of machine learning-based NIDS, and we discuss the limitations of widely used datasets and the challenges they pose for model evaluation and generalisation.

Machine Learning-Based NIDS Architectures

Machine learning-based NIDS have gained prominence for their ability to outperform traditional signature-based methods, offering advantages such as the ability to generalise across different attack scenarios. Several architectures have been proposed, including CNN-based models, autoencoders, and graph neural networks, which aim to detect anomalies and malicious activities in network traffic. For example, Doriguzzi et al. [167], Wang et al. [168] and Tan et al. [169] explore CNNs, while Mirsky et al. [170] adopt an autoencoder-based approach. Additionally, recent work has extended the use of graph

neural networks for NIDS [171, 172]. However, these complex models are often difficult to verify using traditional DNN verification methods. In this work we train fully-connected networks, which achieve high accuracy and are verifiable.

Adversarial Attacks and Defenses for NIDS

Robustness to adversarial examples has emerged as a critical concern in the security of machine learning systems, including NIDS. Zhang et al. [173] provide an extensive review of adversarial attacks and defenses relevant to NIDS, highlighting techniques like HopSkipJump attacks [174] and ensemble adversarial training [175]. The goal of adversarial attacks in NIDS is to subtly manipulate network traffic in ways that cause misclassification by the model. To counter these threats, adversarial training techniques, such as PGD [9], have been employed to make models more resilient. In this work, we employ a customised PGD-based adversarial training approach that focuses on satisfying the desired properties defined by our specifications in Chapter 5.

Dataset Limitations and Generalisation

One of the key challenges in NIDS research is the limited generalisation of models trained and tested on the same public datasets. Often, the datasets used to evaluate NIDS models contain flaws or are overly simplistic [21, 176, 177], leading to models that perform well on the training data but fail to generalise to other similar datasets or different attack scenarios [178, 179]. For instance, research has shown that models trained on benchmark datasets like CIC IDS 2017 and 2018 [22], despite their similarities, often fail when tested on alternative datasets or in cross-dataset evaluations [49, 178, 179]. This issue arises partly because these datasets tend to have predictable patterns of benign traffic, making it easier for models to achieve high performance without genuinely learning useful patterns. Our work aims to address this by generating additional traffic that adheres to predefined specifications and ensuring that models are robust to adversarial inputs within the defined property space.

Probabilistic Verification and Limitations

To the best of our knowledge, no existing research has addressed formal verification for DNN-based NIDS beyond probabilistic approaches. Wang et al. [180] introduced BARS, a framework that provides probabilistic robustness guarantees for DNNs in secu-

rity contexts. BARS employs randomised smoothing, which adds noise to the input data and derives robustness guarantees for the smoothed model. While this approach offers computational efficiency and probabilistic robustness, it has significant limitations. It is non-complete, meaning it only provides probabilistic rather than absolute guarantees, and it fails to incorporate domain-specific constraints that may be crucial for security applications. This limitation is especially pertinent in NIDS, where the input space involves both continuous traffic metrics and discrete features (e.g., protocol types, IP addresses). Our approach, by contrast, defines robustness in a specification-driven manner and employs complete verification, using the frameworks Vehicle [41] and Marabou [33], to ensure that the model adheres to desired properties within a custom-defined subspace.

2.2.8 A Systematic View of Training-Verification Pipelines

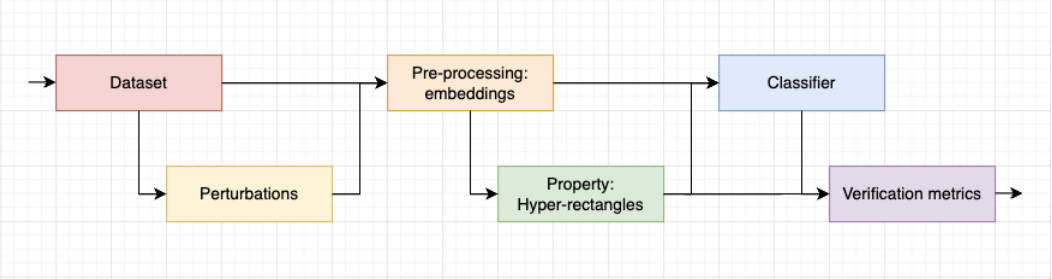


Figure 2.4: *Visualisation of the NLP verification pipeline followed in our approach.*

To systematically compare the existing body of research, we distil an *"NLP verification pipeline"* that is common across many related papers. This pipeline is outlined diagrammatically in Figure 4.1, while Tables 2.2 and 2.3 provide a detailed breakdown, with columns corresponding to each stage. It proceeds in stages:

1. **Given an NLP dataset, generate semantic perturbations on sentences that it contains.** The semantic perturbations can be of different kinds: character, word or sentence level. IBP and randomised smoothing use word and character perturbations, abstract interpretation papers usually do not use any semantic perturbations. Tables 2.2 and 2.3 give the exact mapping of perturbation methods to papers. Our method allows to use all existing semantic perturbations, in particular, we implement character and word level perturbations as in Moradi et al. [126], sentence level perturbations with PolyJuice [129] and Vicuna.

2. **Embed the semantic perturbations into continuous spaces.** The cited papers use

the word embeddings GloVe [125], we use the sentence embeddings S-BERT and S-GPT.

3. **Working on the embedding space, use geometric or semantic perturbations to define geometric or semantic subspaces around perturbed sentences.** In IBP papers, semantic subspaces are defined as "bounds" derived from admissible semantic perturbations. In abstract interpretation papers, geometric subspaces are given by ε-cubes and ε-balls around each embedded sentence. Our work generalises the notion of ε-cubes by defining "hyper-rectangles" on sets of semantic perturbations. The hyper-rectangles generalise ε-cubes both geometrically and semantically, by allowing to analyse subspaces that are drawn around several (embedded) semantic perturbations of the same sentence. We could adapt our methods to work with hyper-ellipses and thus directly generalise ε-balls (the difference boils down to using ℓ_2 norm instead of ℓ_∞ when computing geometric proximity of points), however hyper-rectangles are more efficient to compute, which determined our choice of shapes in this thesis.

4. **Use the geometric/semantic subspaces to train a classifier to be robust to change of label within the given subspaces.** We generally call such training either *robust training* or *semantically robust training*, depending on whether the subspaces it uses are geometric or semantic. A custom semantically robust training algorithm is used in IBP papers, while abstract interpretation papers usually skip this step or use (adversarial) robust training. See Tables 2.2 and 2.3 for further details. In this work, we adapt the famous PGD algorithm [9] that was initially defined for geometric subspaces (ε-balls) to work with semantic subspaces (hyper-rectangles) to obtain a novel semantic training algorithm.

5. **Use the geometric/semantic subspaces to verify the classifier's behaviour within those subspaces.** The papers [132, 133, 134, 135, 136] use IBP algorithms and the papers [137, 138, 139, 140] use abstract interpretation; in both cases it is incomplete and deterministic verification. See 'Verification algorithm' and 'Verification characteristics' columns of Tables 2.2 and 2.3. We use SMT-based tool Marabou (complete and deterministic) and abstract-interpretation tool ERAN (incomplete and deterministic).

Tables 2.2 and 2.3 summarise differences and similarities between existing NLP verifi-

cation approaches and our own. To the best of our knowledge, this is the first work to apply SMT-based complete verification methods in NLP settings. We demonstrate that such methods, particularly those based on the ReLUplex algorithm used by Marabou, offer higher verifiability than prior approaches based on abstract interpretation (e.g., ERAN, CORA) or interval bound propagation and branch-and-bound techniques (e.g., $\alpha\beta$-CROWN), due to their increased precision. While we focus this discussion on NLP, due to its more established verification literature, we later revisit and adapt this pipeline for the NIDS domain in Chapter 5.

Chapter 3

Neural Network Robustness as a Verification Property in Computer Vision

Contents

3.1 Robustness Evaluation in Practice 47

3.2 Relative Comparison of Definitions of Robustness 50

3.2.1 Lipschitz vs Standard Robustness 51

3.2.2 (Strong) Classification Robustness 51

3.2.3 Standard vs Classification Robustness 53

3.2.4 Illustrative Specification in Vehicle 54

3.3 Other Properties of Robustness Definitions 55

3.4 Conclusions 57

In this chapter, we analyse different notions of robustness in the context of computer vision. Robustness is one of the earliest and most studied properties in neural network verification, especially following the discovery of adversarial examples [2, 1]. Despite over a decade of research, the problem remains unsolved: even simple models on simple datasets can exhibit surprising fragility. Core obstacles include the geometric complexity of input spaces, the non-differentiability of logical constraints, and trade-offs between accuracy and robustness during training.

We choose computer vision as our domain of analysis because it removes many confounding factors: inputs are continuous and structured (e.g., pixel matrices), requiring no learned embeddings. The property of robustness itself is also simple in computer vision: it is typically defined using ε-balls around given inputs, without semantic reasoning or symbolic constraints. This makes computer vision an ideal setting to study how different robustness objectives behave during training and verification.

We study four robustness definitions that appear in the verification and adversarial learning literature (see Section 2.1.4):

- Classification Robustness (CR)

- Strong Classification Robustness (SCR)
- Standard Robustness (SR)
- Lipschitz Robustness (LR)

While all are known, their practical differences are often underappreciated. Some properties are strictly stronger than others (e.g., LR implies SR), while others are logically incomparable (e.g., SR vs. SCR). This has important consequences: a system trained for one may not satisfy another, even when the constraints seem intuitively aligned. A key contribution of this chapter is to formally characterise these relationships and demonstrate their impact experimentally.

To evaluate these robustness definitions, we introduce three metrics:

- Constraint Satisfaction (CSat)
- Constraint Accuracy (CAcc)
- Constraint Security (CSec)

Each captures a different notion of robustness quality. CSat checks whether the constraint holds universally; CSec measures how easily the constraint can be violated via attack; and CAcc estimates the empirical frequency of violation under random sampling. These allow for consistent comparisons across different training objectives.

We use standard computer vision datasets (FASHION-MNIST [15] and GTSRB [7]) to empirically test how different training approaches affect robustness and verifiability. We show that some robustness properties are better suited for verification than others, and that aligning the training objective with the verification property is often necessary to achieve meaningful guarantees.

This chapter clarifies the semantics of common robustness objectives and provides a diagnostic framework for detecting mismatches between training and evaluation. In doing so, it sets the stage for later chapters: NLP (Chapter 4), where embeddings and semantic perturbations complicate training and verification, and NIDS (Chapter 5), where inputs are partially discrete and robustness involves global domain constraints.

3.1 Robustness Evaluation in Practice

Having formalised robustness properties and training techniques in Section 2.1.4, we now turn to a key practical question: *how can we evaluate whether a model is truly robust*

in practice?

Despite the appeal of formal verification methods, they are rarely tractable at scale. Moreover, robustness failures can arise even when formal constraints appear satisfied — due to mismatched assumptions between training, verification, and evaluation. This motivates a more nuanced evaluation framework—one that triangulates robustness through verification, adversarial attack, and statistical sampling.

Three Modes of Evaluation

We identify three distinct ways to assess robustness:

1. **Formal Verification**: Does the robustness constraint hold universally within a perturbation region?
2. **Attack Robustness**: How easily can an adversary find a counterexample that violates the constraint?
3. **Sampling Robustness**: What is the empirical likelihood of violation under random perturbation?

Each approach gives rise to a distinct metric, which we define next. Let $\hat{x} \in \mathbb{R}^m$ denote an input point. We define the ε-ball around $\hat{x}$ as:

$$\mathbb{B}(\hat{x}, \varepsilon) := \{x \in \mathbb{R}^m : |\hat{x} - x| \leq \varepsilon\}$$

where $|\cdot - \cdot|$ denotes a distance function (e.g., the ℓ_∞ norm). We also define $\mathbb{I}_{\phi(x)}$ as the standard indicator function, returning 1 if the constraint $\phi(x)$ holds and 0 otherwise.

Constraint Satisfaction (CSat): Formal Verification. Let P denote the target robustness property (e.g., SR, SCR). The *constraint satisfaction* metric checks whether P holds universally within the ε-ball around each test point:

Definition 10 (Constraint Satisfaction)

$$\text{CSat}(\mathscr{D}) = \frac{1}{|\mathscr{D}|} \sum_{\hat{x} \in \mathscr{D}} \mathbb{I}_{\forall x \in \mathbb{B}(\hat{x}, \varepsilon) : P(x)}$$

This corresponds to the strongest form of evaluation — exhaustive, but computationally intractable for larger networks ($< 10^8$ parameters [5]). Note that this definition cor-

responds to verifying local robustness, as it is a guarantee on an ε-ball around a specific input point (see Section 2.1.7).

Constraint Security (CSec): Attack Evaluation. When exact verification is infeasible, a practical alternative is to attempt to break the model using an adversarial attack A. For each $\hat{x}$, the attack searches for a point in $\mathbb{B}(\hat{x}, \varepsilon)$ that violates P:

Definition 11 (Constraint Security)

$$\mathrm{CSec}(\mathscr{D}) = \frac{1}{|\mathscr{D}|} \sum_{\hat{x} \in \mathscr{D}} \mathbb{I}_P(A(\hat{x}))$$

A higher score indicates the model resists attack attempts. In our experiments, A is typically PGD, but any strong attack may be used. Note that CSec depends on both the attack used and the underlying constraint being targeted — e.g., whether the goal is to flip the class (SCR) or exceed a norm bound (SR or LR).

Constraint Accuracy (CAcc): Random Sampling. This metric estimates the probability of a random user coming across a counter-example to the constraint, usually referred as *1 - success rate* in the robustness literature. Let $S(\hat{x}, q)$ be a set of q elements randomly uniformly sampled from $\mathbb{B}(\hat{x}, \varepsilon)$. Then constraint accuracy is defined as:

Definition 12 (Constraint Accuracy)

$$\mathrm{CAcc}(\mathscr{D}) = \frac{1}{|\mathscr{D}|} \sum_{\hat{x} \in \mathscr{D}} \left(\frac{1}{q} \sum_{x \in S(\hat{x}, q)} \mathbb{I}_P(x) \right)$$

While informative in low dimensions, this metric often overestimates robustness in high-dimensional settings, where random samples are unlikely to hit decision boundaries. For example, in our experiments we observe 100% and $> 98\%$ constraint accuracy for SR and SCR, respectively. We will therefore not discuss these experiments in detail.

Why Metrics May Disagree These three metrics often diverge:

- A model may pass random sampling but fail under adversarial attack.
- It may resist known attacks but still violate constraints under formal verification.
- Models trained for one robustness property (e.g., SR) may be evaluated against another (e.g., CR), leading to misleading conclusions.

This motivates a key theme of Chapter 3: *robustness is not monolithic*. The effectiveness of training and the validity of evaluation depend critically on how well the chosen training, verification, and evaluation constraints align.

Choosing a Metric: What Do We Want to Know? Each evaluation mode answers a different question:

- CSat: "Is the model provably robust?"
- CSec: "Can an attacker break it?"
- CAcc: "Is it robust in practice, on average?"

We advocate reporting all three metrics when feasible, and explicitly stating the underlying constraint P being evaluated. The choice of constraint (e.g., SR vs SCR) directly influences how adversarial risk is measured and mitigated.

We will use this evaluation framework to experimentally study mismatches between training and evaluation setups — a key cause of failure in practice.

3.2 Relative Comparison of Definitions of Robustness

We now compare the robustness properties introduced in Section 2.1.4 using the evaluation metrics from Section 3.1. Our goal is to understand how these properties relate both theoretically and empirically, and how training for one objective affects a network's performance under others.

We train networks on the *FASHION MNIST* (or just *FASHION*) [15] and a modified version of the *GTSRB* [7] datasets, using 28×28 and 48×48 greyscale images, respectively. Each dataset has 10 classes. The model architecture is a two-layer fully connected network with 100 ReLU units in the hidden layer and 10 linear outputs, clamped to $[-100, 100]$ for compatibility with verifiers such as Marabou [33], which do not support softmax.

We train four models per dataset: a baseline model (standard loss), and models trained for SR, LR, and SCR. This yields eight networks in total. While we typically show results from one dataset per figure, qualitative trends are consistent across both.

3.2.1 Lipschitz vs Standard Robustness

Lipschitz robustness is strictly stronger than standard robustness: if a model satisfies $LR(\varepsilon, L)$, it also satisfies $SR(\varepsilon, \varepsilon L)$. The reverse does not hold: SR places no constraint on the relation between input and output distances, so any fixed L can be violated with sufficiently small $|x - \hat{x}|$.

Empirical significance of the conclusions for constraint security. Figure 3.1 shows that LR-trained networks achieve higher robustness than SR-trained ones, even under SR-style attacks. This confirms the theoretical hierarchy and illustrates a practical insight: stronger constraints produce more robust models, though they are harder to satisfy during training.

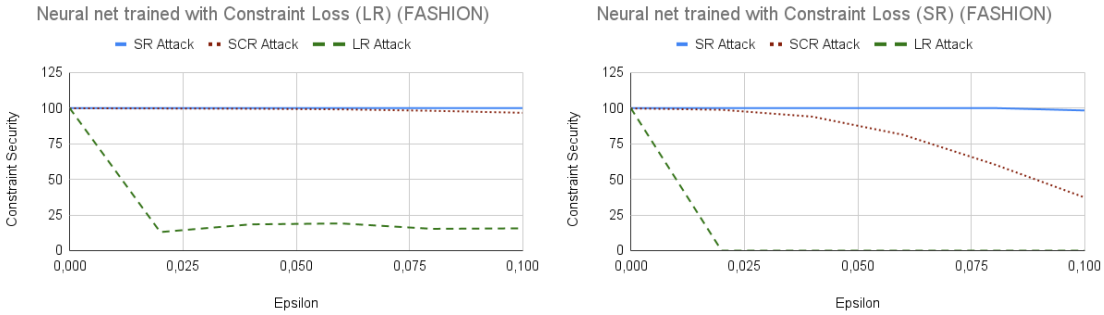


Figure 3.1: *Constraint security of SR and LR-trained networks against PGD attacks using different robustness definitions. ε denotes attack strength.*

Empirical significance of the conclusions for constraint satisfaction. Table 3.1 shows that LR is very difficult to guarantee as a verification property, indeed none of our networks satisfied this constraint for any image in the data set. At the same time, networks trained with LR satisfy the weaker property SR, for 100% and 97% of images – a huge improvement on the negligible percentage of robust images for the baseline network. Therefore, knowing a verification property or mode of attack, one can tailor the training accordingly, and training with stronger constraint gives better results.

3.2.2 (Strong) Classification Robustness

Strong classification robustness is designed to over-approximate classification robustness whilst providing a logical loss function with a meaningful gradient. We work under the assumption that the last layer of the classification network is a softmax layer, and

	FASHION net trained with:				GTSRB net trained with:			
	Baseline	**SCR**	**SR**	**LR**	**Baseline**	**SCR**	**SR**	**LR**
CR satisfaction	1.5	2.0	2.0	34.0	0.5	1.0	3.0	4.5
SR satisfaction	0.5	1.0	65.8	100.0	0.0	0.0	24.0	97.0
LR satisfaction	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 3.1: *Constraint satisfaction results for the CR, SR, and LR constraints. Calculated over the test set and expressed as percentages.*

therefore the output forms a probability distribution. When $\eta > 0.5$ then any network that satisfies $SCR(\varepsilon, \eta)$ also satisfies $CR(\varepsilon)$. For $\eta \leq 0.5$ this relationship breaks down as the true class may be assigned a probability greater than η but may still not be the class with the highest probability. We therefore recommended that one only uses value of $\eta > 0.5$ when using strong classification robustness (for example $\eta = 0.52$ in [30]).

Empirical significance of the conclusions for constraint security. We train models using SCR and compare their robustness to those trained via data augmentation (a proxy for CR). Figure 3.2 shows that data augmentation performs poorly compared to adversarial training or constraint-based losses, especially on smaller networks. Interestingly, even baseline models outperform data-augmented ones. However, by replicating the results with a larger, 18-layer convolutional network [30], we confirm that larger networks handle data augmentation better, and that data augmentation yields improved robustness compared to the baseline.

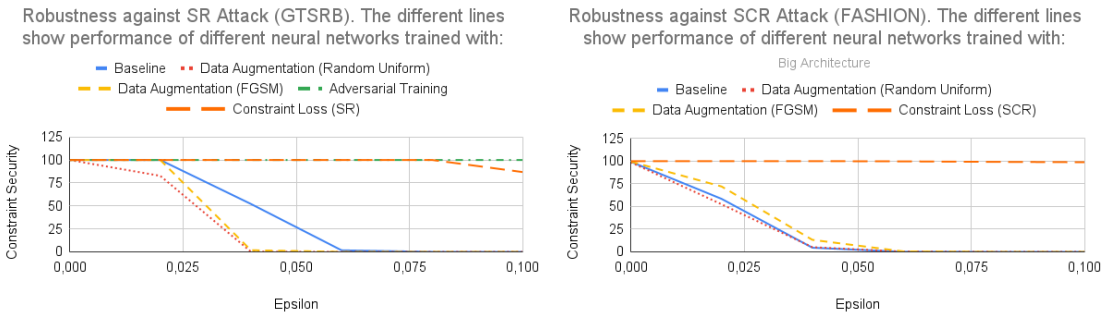


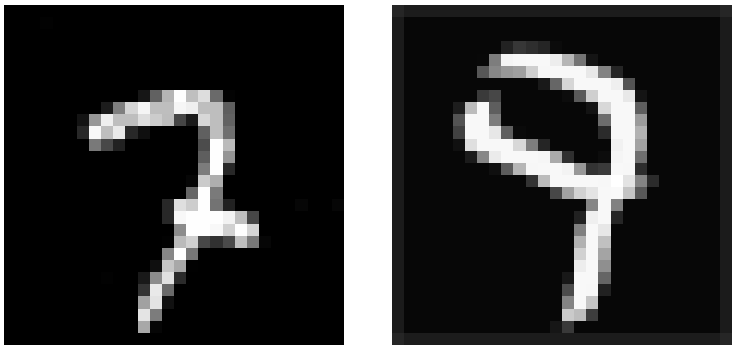
Figure 3.2: *Experiments that show how adversarial training, training with data augmentation, and training with constraint loss affect standard and classification robustness of networks; ε measures the attack strength.*

Empirical significance of the conclusions for constraint satisfaction. Although Table 3.1 confirms that training with a stronger property (SCR) does improve the constraint satisfaction of a weaker property (CR), the effect is an order of magnitude smaller than what we observed for LR and SR. Indeed, the table suggests that training with the

LR constraint gives better results for CR constraint satisfaction. This does not contradict, but does not follow from our theoretical analysis.

3.2.3 Standard vs Classification Robustness

Given that LR is stronger than SR and SCR is stronger than CR, the obvious question is whether there is a relationship between these two groups. In short, the answer to this question is no. In particular, although the two sets of definitions agree on whether a network is robust around images with high-confidence, they disagree over whether a network is robust around images with low confidence. We illustrate this with an example, comparing SR against CR. We note that a similar analysis holds for any pairing from the two groups.



(a) $P(7) = 85\%$ (b) $P(7) = 51\%$

Figure 3.3: *Images from MNIST: (left) high confidence; (right) low confidence.*

The key insight is that standard robustness bounds the drop in confidence that a neural network can exhibit after a perturbation, whereas classification robustness does not. Fig. 3.3 shows two hypothetical images from the MNIST dataset. Our network predicts that Fig. 3.3a has an 85% chance of being a 7. Now consider adding a small perturbation to the image and consider two different scenarios. In the first scenario the output of the network for class 7 decreases from 85% to 83% and therefore the classification stays the same. In the second scenario the output of the network for class 7 decreases from 85% to 45%, and results in the classification changing from 7 to 9. When considering the two definitions, a small change in the output leads to no change in the classification and a large change in the output leads to a change in classification and so standard robustness and classification robustness both agree with each other.

However, now consider Fig. 3.3b with relatively high uncertainty. In this case the network is (correctly) less sure about the image, only narrowly deciding that it's a 7. Again consider adding a small perturbation. In the first scenario the prediction of the

network changes dramatically with the probability of it being a 7 increasing from 51% to 91% but leaves the classification unchanged as 7. In the second scenario the output of the network only changes very slightly, decreasing from 51% to 49% flipping the classification from 7 to 9. Now, the definitions of SR and CR disagree. In the first case, adding a small amount of noise has erroneously massively increased the network's confidence and therefore the SR definition correctly identifies that this is a problem. In contrast CR has no problem with this massive increase in confidence as the chosen output class remains unchanged. Thus, SR and CR agree on low-uncertainty examples, but CR breaks down and gives what we argue are both false positives and false negatives when considering examples with high-uncertainty.

Empirical significance of the conclusions for constraint security. Our empirical study confirms these general conclusions. Fig. 3.1 shows that depending on the properties of the dataset, SR may not guarantee SCR. The results in Fig. 3.4 tell us that using the SCR constraint for training does not help to increase defences against SR attacks. A similar picture, but in reverse, can be seen when we optimize for SR but attack with SCR. Table 3.1 confirms these trends for constraint satisfaction. This illustrates the importance of matching the training and evaluation constraint definitions.

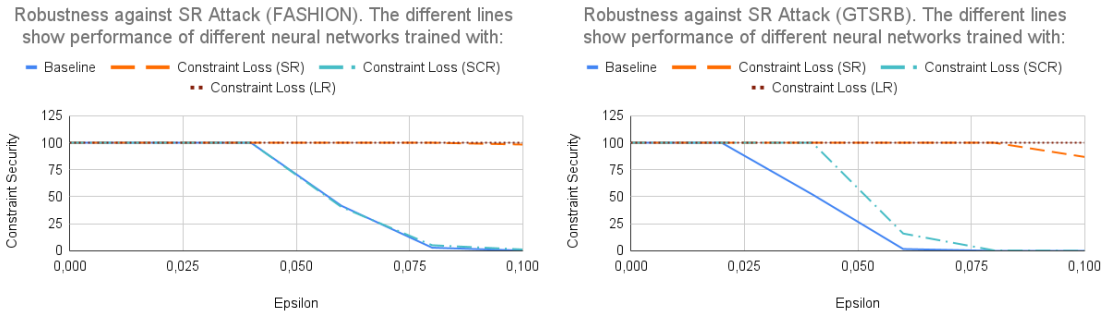


Figure 3.4: *Experiments that show how different choices of a constraint loss affect standard robustness of neural networks.*

3.2.4 Illustrative Specification in Vehicle

Although our computer vision experiments do not use Vehicle directly, many robustness constraints can be concisely expressed using specification languages such as Vehicle. For clarity and consistency, we use it throughout the thesis to illustrate how robustness or constraint properties can be encoded. Figure 3.5 shows an example of how classification robustness $CR(\varepsilon)$ could be specified over FASHION images. Note how lines 9–10 define

a `validImage` check — in Chapter 5, we implement a similar method to enforce admissible input constraints in the NIDS setting. Lines 12–13 define the `advises` predicate, which specifies the most confident classification label; this predicate recurs across multiple domains for encoding classification-based properties. See Section 2.1.6 for a broader discussion of specification languages.

```
1  -- Illustrative Vehicle spec for computer vision
2
3  @network
4  classifier : Tensor Rat [28, 28] -> Vector Rat 10
5
6  @parameter
7  epsilon : Rat
8
9  validImage : Tensor Rat [28, 28] -> Bool
10 validImage x = forall i j . 0 <= x ! i ! j <= 1
11
12 advises : Tensor Rat [28, 28] -> Index 10 -> Bool
13 advises x i = forall j . j != i => classifier x ! i > classifier x ! j
14
15 boundedByEpsilon : Image -> Bool
16 boundedByEpsilon x = forall i j . -epsilon <= x ! i ! j <= epsilon
17
18 robustAround : Image -> Label -> Bool
19 robustAround image label = forall pertubation .
20   let perturbedImage = image - pertubation in
21   boundedByEpsilon pertubation and validImage perturbedImage =>
22     advises perturbedImage label
23
24 @parameter(infer=True)
25 n : Nat
26
27 @dataset
28 trainingImages : Vector Image n
29
30 @dataset
31 trainingLabels : Vector Label n
32
33 @property
34 robust : Vector Bool n
35 robust = foreach i . robustAround (trainingImages ! i) (trainingLabels ! i)
```

Figure 3.5: *An illustrative Vehicle specification for CR(ε). This was not used in practice but shows how such a property could be encoded formally.*

3.3 Other Properties of Robustness Definitions

We conclude this chapter by examining higher-level considerations that affect the selection and application of robustness definitions in practice. These include assumptions about the data distribution, the interpretability of different constraints, and their alignment with common training methods. Table 3.2 summarises key distinctions between the four definitions discussed.

Definition	Standard robustness	Lipschitz robustness	Classification robustness	Strong class. robustness
Problem domain	General	General	Classification	Classification
Interpretability	Medium	Low	High	Medium
Globally desirable	✓	✓	×	×
Has loss functions	✓	✓	×	✓
Adversarial training	✓	×	✓	×
Data augmentation	×	×	✓	×
Logical-constraint training [30]	✓	✓	×	✓

Table 3.2: *A comparison of the different types of robustness studied in this work. Top half: general properties. Bottom half: relation to existing machine-learning literature*

Dataset Assumptions. Dataset assumptions concern the relationship between training data and the true data manifold. These assumptions influence both the definition and evaluation of robustness. For SR and LR it is, at minimum, desirable for the network to be robust over the entire data manifold. Since the shape of the manifold is typically unknown, it is approximated by taking the union of ε-balls around training inputs. We are not particularly interested about whether the network is robust in regions of the input space that lie off the data manifold, but there is no problem if the network is robust in these regions. Therefore these definitions make no assumptions about the distribution of the training dataset.

In contrast, CR and SCR make stronger assumptions. They enforce that the classification should not change under perturbation, which is only meaningful if the perturbation region does not cross a decision boundary. Consequently when one is training for some form of classification robustness, one is implicitly making the assumption that the training data points lie away from any decision boundaries within the manifold. In practice, most datasets for classification problems assign a single label instead of an entire probability distribution to each input point, and so this assumption is usually valid. However, for datasets that contain input points that may lie close to the decision boundaries, CR and SCR may result in a logically inconsistent specification.

Interpretability of Robustness Constraints. One of the key selling points of training with logical constraints is that, by ensuring that the network obeys understandable constraints, it improves the explainability of the neural network. Each of the robustness constraints encode that "small changes to the input only result in small changes to the output", but the interpretability of each definition is also important.

All definitions use the parameter ε to define the size of perturbation regions. This is

generally intuitive, as it directly specifies the notion of a "small change" in input space. CR is arguably the most interpretable property: it requires that the predicted class remains unchanged under perturbation, with no additional parameters. This simplicity aligns well with user expectations and is easy to communicate. In contrast, SR and SCR introduce secondary parameters: δ for SR (bounding output deviation) and η for SCR (a softmax confidence threshold). These weaken interpretability somewhat, as their effect depends on the output scale and may be non-obvious to users.

LR is the most technically demanding and least interpretable. Its parameter L sets a bound on how fast the output can change relative to the input. This involves comparing quantities in very different spaces—e.g., pixel distances and probability distributions. The resulting ratio (e.g., "output changes no more than L times the input change") often lacks intuitive meaning. For example, allowing the KL divergence between outputs to grow twice as fast as the ℓ_∞ norm of pixel perturbations is difficult to interpret semantically. This illustrates a general trade-off: stronger, more complex constraints tend to be harder to interpret and apply in practice.

3.4 Conclusions

In this chapter, we demonstrated the importance of studying robustness properties of neural networks independently of specific training methods. By treating these properties as mathematical abstractions, we were able to reason about their logical relationships, empirical implications, and practical usability. We showed that some robustness definitions are logically ordered by strength (e.g., LR implies SR; SCR implies CR), while others are incomparable (e.g., SR vs. CR). When such orderings exist, training for a stronger property often improves verification outcomes for a weaker one. However, stronger properties such as LR remain challenging to verify with current solvers like Marabou [33]. Importantly, logical strength does not guarantee other desirable traits such as interpretability. For instance, CR is easier to understand than LR, despite being logically weaker.

These findings support several recommendations:

- Avoid CR and SCR where possible, due to their limited interpretability and problematic behaviour near decision boundaries.
- Prefer training for stronger properties (e.g., LR) when aiming to verify a weaker one (e.g., SR).

- Interpret SR and CR with caution in ambiguous regions of the input space, as their behaviours may diverge.

We also introduced and analysed three evaluation metrics—constraint satisfaction, constraint security, and constraint accuracy—each of which captures a different aspect of robustness. Among these, constraint security (CSec) strikes a practical balance: it is more informative than random sampling (CAcc), more scalable than formal verification (CSat), and effective for guiding training and evaluation.

Overall, this chapter provides a methodology for analysing robustness definitions and their trade-offs within a continuous verification pipeline. By decoupling the semantics of robustness from specific implementations, we aim to bridge the gap between formal verification and practical machine learning workflows.

Chapter 4

Neural Network Robustness as a Verification Property in NLP

Contents

4.1 The Problem of NLP Verification 61

4.2 The Parametric NLP Verification Pipeline 62

4.2.1 Datasets 63

4.2.2 Semantic Perturbations 64

4.2.3 NLP Embeddings 66

4.2.4 Working with Embedding Spaces: Our Approach 67

4.2.5 Training 72

4.2.6 Choice of Verification Algorithm 73

4.3 Characterisation of Verifiable Subspaces 74

4.3.1 Metrics for Understanding the Properties of Embedding Spaces 75

4.3.2 Baseline Experiments for Understanding the Properties of Embedding Spaces 76

4.3.3 Verifiability-Generalisability Trade-off for Geometric Subspaces 77

4.3.4 Verifiability-Generalisability Trade-off for Semantic Subspaces 80

4.3.5 Adversarial Training on Semantic Subspaces 83

4.4 NLP Case Studies 87

4.4.1 Role of False Positives and False Negatives 89

4.4.2 Performance of Existing LLMs as Safety-Critical Filters . . . 90

4.4.3 Experimental Setup of the Verification Pipeline 91

4.4.4 Analysis of the Role of Embedding Functions 93

4.4.5 Analysis of Perturbations 95

4.4.6 Embedding Error 103

4.5 Conclusions and Future Work 106

This chapter extends the robustness framework introduced in Chapter 3 to the domain of natural language processing. While robustness in computer vision can often be characterised by geometric perturbations (e.g., ℓ_p-bounded changes in pixel space), NLP presents a fundamentally different challenge: inputs are discrete, highly structured, and sensitive to even small changes. For example, replacing a word or altering a character can yield substantial shifts in meaning, making naive perturbation-based definitions of robustness unsuitable.

At the same time, modern NLP systems frequently rely on deep neural networks applied to continuous vector embeddings, enabling — at least in theory — the use of formal verification techniques developed for continuous domains. This opens the door to verifying robustness properties over embedding space.

In this chapter, we revisit the family of properties studied in Chapter 3 through the more general lens of *Classification Invariance (CI)*. We consider both *local* regions—ℓ_∞ balls ("ε-cubes") around a reference sentence—and *global* hyper-rectangles built from paraphrase sets in the embedding space. CI strictly generalises *Classification Robustness (CR)*: when the region is a norm ball, CI reduces to CR; otherwise it captures invariance over arbitrary (axis-aligned) regions. While Chapter 3 recommends stronger or more interpretable notions when possible, the geometry of NLP inputs justifies CI: hyper-rectangular subspaces allow each dimension to vary independently, making notions such as SR and LR—which assume uniform input perturbations across all dimensions—harder to interpret or enforce. CI, by contrast, remains meaningful: it requires that all points within a given hyper-rectangular subspace be assigned the correct class. We formalise this as *Classification Invariance* (Definition 14) and use it for both training and verification.

A central complication is the *embedding gap* — the disconnect between symbolic linguistic inputs and their continuous embeddings. Formal verification operates in continuous space, but its guarantees may not translate to meaningful statements about real sentences. This mismatch means that a network might be verifiable over a given subspace of embeddings, while the corresponding region contains no interpretable or semantically coherent language. This chapter formalises this problem and proposes a principled verification methodology that works within embedding space while remaining sensitive to semantic structure in language.

This chapter contributes a principled methodology for CI/CR-based robustness veri-

fication in NLP. We:

1. Formalise verification in embedding space and identify key obstacles due to the embedding gap.

2. Define a variety of embedding subspaces and introduce metrics for generalisability and semantic fidelity.

3. Develop robust training techniques that align geometric and semantic robustness in NLP.

4. Evaluate verification and training performance across different perturbation types and datasets.

Our results show that it is possible to construct verification pipelines that retain the modularity of vision-based approaches, but adapted to the constraints of language. The remainder of the chapter is organised as follows: Section 4.1 formalises the verification problem; Section 4.2 presents our training and verification methodology; Section 4.3 defines subspaces and generalisability metrics; and Section 4.4 evaluates experimental performance and implications.

4.1 The Problem of NLP Verification

Formal verification techniques for deep neural networks aim to formally guarantee that, for every possible input within a defined region of the input space, the network satisfies a specified property. As previously mentioned, in computer vision tasks, such regions are typically ε-balls around input images, and verification tools can reason over continuous perturbations to ensure robustness. Applying the same methodology to NLP is far more challenging due to the discrete nature of linguistic inputs and the non-invertibility of embedding functions.

Despite these challenges, the use of embeddings makes robustness verification in NLP feasible, at least in principle, by treating verification as a problem over subspaces of the continuous embedding space. Concretely, given a network $N : \mathbb{R}^m \to \mathbb{R}^n$, a sentence s and an embedding function $E : s \to \mathbb{R}^m$, one can define subspaces $\mathscr{S}_1, \ldots, \mathscr{S}_l$ around the embedded representations of an NLP dataset $\mathscr{Y}$ of natural language sentences $s_1, \ldots, s_q$. These subspaces may take the form of ε-balls, hyper-rectangles, or convex hulls, and are typically centred on the embeddings of specific sentences.

A verification algorithm $\mathscr{V}$ then determines whether N is robust on each subspace $\mathscr{S}_i$, i.e whether the network assigns the same class label to every point in $\mathscr{S}_i$. As each subspace is infinite (due to the continuity of $\mathbb{R}^m$), verification is usually based on equational reasoning, abstract interpretation or bound propagation (see related work in Section 2.2.5). The proportion of subspaces for which robustness can be formally verified defines the network's *verification success rate*, or its *verifiability*. When comparing two networks, we say that N_1 is more verifiable than N_2 if its success rate is higher on the same set of subspaces.

However, applying this approach in NLP faces a central obstacle: the *embedding gap*. The embedding function E is not surjective: most vectors in $\mathbb{R}^m$ do not correspond to any valid sentence. This means that even if robustness is verified over a subspace of embedding space, it may say little about how the model behaves on actual linguistic inputs. Worse, the subspace may contain no interpretable sentences at all, rendering verification vacuous. We refer to such cases as having *low generalisability*. Conversely, even when a verified subspace contains many interpretable sentences, semantically incorrect sentences may be incorrectly included, resulting in *embedding errors*, where unrelated inputs are treated as equivalent due to geometric proximity in the embedding space.

Verification in NLP must therefore consider both verifiability and generalisability. A model may be provably robust over large regions of embedding space, but this only has practical value if those regions correspond to meaningful, semantically coherent sets of sentences. Bridging this gap between geometric robustness and linguistic relevance is the central aim of this chapter.

We proceed by formalising subspace definitions and metrics for generalisability in Section 4.2, followed by a comprehensive evaluation of semantic versus geometric subspaces in Section 4.3. Our training methods, introduced in Section 4.2.5, are tailored to improve robustness in these subspaces. Together, these components form a modular verification pipeline that connects insights from NLP, machine learning, and formal verification.

4.2 The Parametric NLP Verification Pipeline

This section presents a *parametric NLP verification* pipeline, shown in Figure 4.1 diagrammatically. We call it "parametric" because each component within the pipeline is defined independently of the others and can be taken as a parameter when studying other

components. The parametric nature of the pipeline allows for the seamless integration of state-of-the-art methods at every stage, and for more sophisticated experiments with those methods. The following section provides a detailed exposition of the methodological choices made at each step of the pipeline.

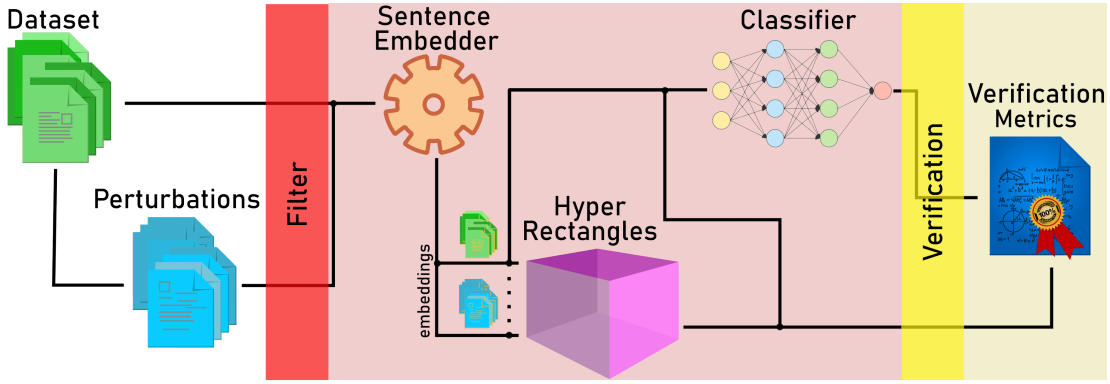


Figure 4.1: *Visualisation of the NLP verification pipeline followed in our approach.*

4.2.1 Datasets

This work focuses on safety-critical NLP applications where incorrect outputs may have regulatory, legal, or ethical consequences. Accordingly, we evaluate our verification pipeline on two datasets that model real-world compliance scenarios: chatbot identity disclosure and medical safety.

R-U-A-Robot [19]. This dataset is motivated by legislation and regulatory proposals (e.g., [160, 161, 159]) requiring artificial systems, such as chatbots, to truthfully disclose their non-human identity. It contains 6,800 written English utterances related to the intent *"Are you a robot?"*, including variations such as "I'm a man, what about you?" and "You're not human, right?". The data was collected using grammar templates, crowdsourcing, and pre-existing corpora.

The dataset consists of 2,720 positive examples (where identity disclosure is appropriate), 3,400 negative examples (where it is not), and 680 ambiguous examples (where it is unclear whether the system is required to state its identity). Given safety-critical priorities, we collapse the ambiguous and positive classes into a single *positive* label to favour recall over precision. The result is a binary classification task aimed at minimising false negatives.

This dataset is particularly relevant in light of concerns about user deception in systems like Google Duplex [181], which faced criticism for producing human-sounding

outputs [182]. As such, R-U-A-Robot reflects an emerging need for verifiable identity disclosure in deployed AI systems [183].

Medical Safety [20]. In medical contexts, NLP systems that respond to high-risk user queries without qualification can cause harm [162]. For example, a system may appear authoritative while providing medically dangerous advice [163]. The Medical Safety dataset contains 2,917 user queries (1,417 medical, 1,500 non-medical), collected from online forums such as Reddit's `r/AskDocs`.

Queries are annotated by experts and crowd-workers for both domain relevance and risk level, following the World Economic Forum (WEF) taxonomy of chatbot risk [184]. To enable effective training and verification, we merge all medically relevant queries into a single *positive* class, resulting in a binary classification task distinguishing medical versus non-medical content. We use only those queries for which expert consensus on domain label was achieved.

This dataset will facilitate discussion on how to guarantee that a system recognises medical queries, in order to avoid generating medical output.

Semantic Distinctions. These two datasets were selected not only for their societal relevance, but also for their differing semantic properties. R-U-A-Robot contains many semantically similar but lexically different queries, while Medical Safety includes semantically diverse queries. This contrast supports evaluation of robustness and generalisation across different linguistic phenomena.

For both datasets, we use the original train/test splits provided in their source papers. The *positive* label indicates inputs requiring a safety-critical response (disclosure or medical awareness), and our verification pipeline is designed to assess whether such cases are correctly handled under perturbations and model uncertainty.

4.2.2 Semantic Perturbations

As discussed in Section 2.2.5, we require semantic perturbations for creating semantic subspaces. To do so, we consider three *kinds* of perturbations – i.e. character, word and sentence level. This systematically accounts for different variations of the samples.

Character and word level perturbations are created via a rule-based method proposed by Moradi et al. [126] to simulate different kinds of noise one could expect from spelling mistakes, typos etc. These perturbations are non-adversarial and can be generated auto-

matically. Moradi et al. [126] found that NLP models are sensitive to such small errors, while in practice this should not be the case. Character level perturbations *types* include randomly inserting, deleting, replacing, swapping or repeating a character of the data sample. At the character level, we do not apply letter case changing, given it does not change the sentence-level representation of the sample. Nor do we apply perturbations to commonly misspelled words, given only a small percentage of the most commonly misspelled words occur in our datasets. Perturbations types at the word level include randomly repeating or deleting a word, changing the ordering of the words, the verb tense, singular verbs to plural verbs or adding negation to the data sample. At the word level, we omit replacement with synonyms, as this is accounted for via sentence rephrasing. Negation is not done on the medical safety dataset, as it creates label ambiguities (e.g. 'pain when straightening knee' $\rightarrow$ 'no pain when straightening knee'), as well as singular plural tense and verb tense, given human annotators would experience difficulties with this task (e.g. rephrase the following in plural/ with changed tense – 'peritonsillar abscess drainage aftercare.. please help'). Note that the Medical dataset contains several sentences without a verb (like the one above) for which it is impossible to pluralise or change the tense of the verb.

Further examples of character and word rule-based perturbations can be found in Tables 4.1 and 4.2.

Method	**Description**	**Altered sentence** (*Are you a robot?*)
Insertion	A character is randomly selected and inserted in a random position.	*Are yovu a robot?*
Deletion	A character is randomly selected and deleted.	*Are you a robt?*
Replacement	A character is randomly selected and replaced by an adjacent character on the keyboard.	*Are you a ronot?*
Swapping	A character is randomly selected and swapped with the adjacent right or left character in the word.	*Are you a rboot?*
Repetition	A character in a random position is selected and duplicated.	*Arre you a robot?*

Table 4.1: *Character-level perturbations: their types and examples of how each type acts on a given sentence from the R-U-A-Robot dataset [19]. Perturbations are selected from random words that have 3 or more characters, first and last characters of a word are never perturbed.*

Sentence level perturbations. We experiment with two types of sentence level perturbations, particularly due to the complicated nature of the medical queries (e.g. it is non-trivial to rephrase queries such as this – 'peritonsillar abscess drainage aftercare.. please help'). We do so by either using Polyjuice [129] or `vicuna-13b`[1]. Polyjuice is a

[1]Using the following API: `https://replicate.com/replicate/vicuna-13b/api`.

Method	**Description**	**Altered sentence** (*Can u tell me if you are a chatbot?*)
Deletion	Randomly selects a word and removes it.	*Can u tell if you are a chatbot?*
Repetition	Randomly selects a word and duplicates it.	*Can can u tell me if you are a chatbot?*
Negation	Identifies verbs then flips them (negative/positive).	*Can u tell me if you are not a chatbot?*
Singular/ plural verbs	Changes verbs to singular form, and conversely.	*Can u tell me if you is a chatbot?*
Word order	Randomly selects consecutive words and changes the order in which they appear.	*Can u tell me if you are chatbot a?*
Verb tense	Converts present simple or continuous verbs to their corresponding past simple or continuous form.	*Can u tell me if you were a chatbot?*

Table 4.2: *Word-level perturbations: their types and examples of how each type acts on a given sentence from the R-U-A-Robot dataset [19].*

general-purpose counterfactual generator that allows for control over perturbation types and locations, trained by fine-tuning GPT-2 on multiple datasets of paired sentences. Vicuna is a state-of-the-art open source chatbot trained by fine-tuning LLaMA [185] on user-shared conversations collected from ShareGPT [2]. For Vicuna, we use the following prompt to generate variations on our data samples '*Rephrase this sentence 5 times: "[Example]"*.' For example, from the sentence "How long will I be contagious?", we can obtain "How many years will I be contagious?" or "Will I be contagious for long?" and so on.

We will use notation $\mathscr{P}$ to refer to a perturbation algorithm abstractly.

Semantic similarity of perturbations. In later sections we will make an assumption that the perturbations that we use produce sentences that are semantically similar to the originals. However, precisely defining or measuring semantic similarity is a challenge in its own right, as semantic meaning of sentences can be subjective, context-dependent, which makes evaluating their similarity intractable. Nevertheless, Subsection 4.4.5 will discuss and use several metrics for calculating semantic similarity of sentences, modulo some simplifying assumptions.

4.2.3 NLP Embeddings

The next component of the pipeline is the embeddings. As discussed in Section 2.1.1, neural embeddings are a key mechanism for mapping discrete textual data into continuous vector spaces. This transformation enables the application of gradient-based training and verification techniques to NLP tasks.

In our verification pipeline, the embedding function $E : \mathbb{S} \to \mathbb{R}^m$ is implemented using

[2] https://sharegpt.com/

pre-trained transformer models, such as Sentence-BERT [17] and Sentence-GPT [18]. These models approximate semantic similarity between sentences by placing semantically related inputs closer together in the embedding space $\mathbb{R}^m$.

However, as introduced in Section 4.1, this embedding space suffers from a known limitation: not every point in $\mathbb{R}^m$ corresponds to a valid sentence. This non-surjectivity [6], which plays a crucial role in the *embedding gap*, complicates formal verification and motivates the careful construction of interpretable subspaces and robustness specifications.

4.2.4 Working with Embedding Spaces: Our Approach

Building on the types of embedding spaces introduced in Section 2.1.5, we now define specific subspaces used for verification in NLP. In particular, we formalise the construction of geometric and semantic subspaces that support robustness training and verification. Our goal is to define subspaces in the embedding space $\mathbb{R}^m$ through effective algorithmic procedures. We use the notation $\mathscr{S}$ to refer to a subspace of the embedding space. Recall that a hyper-rectangle of dimension m is a list of intervals $(a_1,b_1),\ldots,(a_m,b_m)$ such that a point $x \in \mathbb{R}^m$ is a member if for every dimension j we have $a_j \leq x_j \leq b_j$.

We start with an observation that, given an NLP dataset $\mathscr{Y}$ that contains a finite set of sentences $s_1,\ldots,s_q$ belonging to the same class, and an embedding function $E : \mathbb{S} \to \mathbb{R}^m$, we can define an *embedding matrix* $\mathscr{X} \in \mathbb{R}^{q\times m}$, where each row j is given by $E(s_j)$. We will use the notation x_i to refer to the ith element of the vector x, and $\mathscr{X}^{ij}$ to refer to the element in the ith row and jth column of $\mathscr{X}$. Treating embedded sentences as matrices, rather than as points in the real vector space, makes many computations easier. We can therefore define a *hyper-rectangle* for $\mathscr{X}$ as follows.

Definition 13 (Hyper-rectangle for an Embedding Matrix) *Given an embedding matrix $\mathscr{X} \in \mathbb{R}^{q\times m}$, the m-dimensional* hyper-rectangle *for $\mathscr{X}$ is defined as:*

$$\mathbb{H}(\mathscr{X}) := \{(\min_{i=0}^{q} \mathscr{X}^{ij},\ \max_{i=0}^{q} \mathscr{X}^{ij}) \mid j \in [1,...,m]\}$$

Therefore given an embedding function $E : \mathbb{S} \to \mathbb{R}^m$, and a set of sentences $\mathscr{Y} = \{s_1,\ldots,s_q\}$, we can form a subspace $\mathbb{H}(E(\mathscr{Y}))$ by constructing the embedding matrix, as described above, and forming the corresponding hyper-rectangle. To simplify the notation, we will omit the application of E and from here on simply write $\mathbb{H}(\mathscr{Y})$.

The next example shows how the above definitions generalise the commonly known definition of the ε-cube.

Example 4 (ε-cube and ε-ball) *One of the most popular terms used in robust training [1] and verification [50] literature is the* ε-ball. *It is defined as follows. Given an embedded input* $\hat{x}$*, a constant* $\varepsilon \in \mathbb{R}$*, and a distance function (*ℓ*-norm)* $|-|$*, the* ε-ball *around* $\hat{x}$ *of radius* ε *is defined as:*

$$\mathbb{B}(\hat{x}, \varepsilon) := \{x \in \mathbb{R}^m : |\hat{x} - x| \leq \varepsilon\}.$$

In practice, it is common to use the ℓ_∞ norm, which results in the ε-ball actually being a hyper-rectangle, also called ε-*cube*, where $(a_j, b_j) = (\hat{x}_j - \varepsilon, \hat{x}_j + \varepsilon)$. Therefore our construction $\mathbb{H}$ is a strict generalisation of ε-cubes. We will therefore use the notation $\mathbb{H}(\mathscr{Y}, \varepsilon) = \bigcup_{s \in \mathscr{Y}} \mathbb{B}(E(s), \varepsilon)$ to refer to the set of ε-cubes around every sentence in the dataset.

Building on the above, we generalise *Classification Robustness* (Def. 2) to any hyper-rectangle as follows.

Definition 14 (Classification Invariance) *Let* $\mathbb{H}$ *be a hyper-rectangle and let* c *be the intended class for its points:*

$$CI(\mathbb{H}) := \forall x \in \mathbb{H} \Rightarrow \arg\max N(x) = c$$

Definition 14 recovers the Classification Robustness (CR) when $\mathbb{H}$ is an ε-cube and strictly generalises it otherwise. In the same spirit, one could generalise *standard robustness* by replacing the invariance requirement accordingly; we omit those details here, as we do not use them in this chapter.

Of course, as previously discussed and illustrated in Figure 2.1, hyper-rectangles are not very precise, geometrically. A more precise shape would be a *convex hull* around q given points in the embedding space. Indeed literature has some definitions of convex hulls [186, 187, 188]. However, none of them is suitable as they are computationally too expensive due to the time complexity of $O(q^{m/2})$ where q is the number of inputs and m is the number of dimensions [186]. Approaches that use under-approximations to speed up the algorithms [187, 188] do not work well in NLP scenarios, as under-approximated subspaces are so small that they contain near zero sentence embeddings.

Exclusion of Unwanted Sentences Via Shrinking

Another concern is that the generated hyper-rectangles may contain sentences from a different class. This would make it unsuitable for verification. In order to exclude all samples from the wrong class, we define a shrinking algorithm $SH(\mathscr{X}, \mathscr{Y}, c)$ that calculates a new subspace that is a subset of the original hyper-rectangle around $\mathscr{X}$, that only contains embeddings of sentences in $\mathscr{Y}$ that are of class c. Of course, to ensure this, the algorithm may have to exclude some sentences of class c. The second graph of Figure 4.2 gives a visual intuition of how this is done.

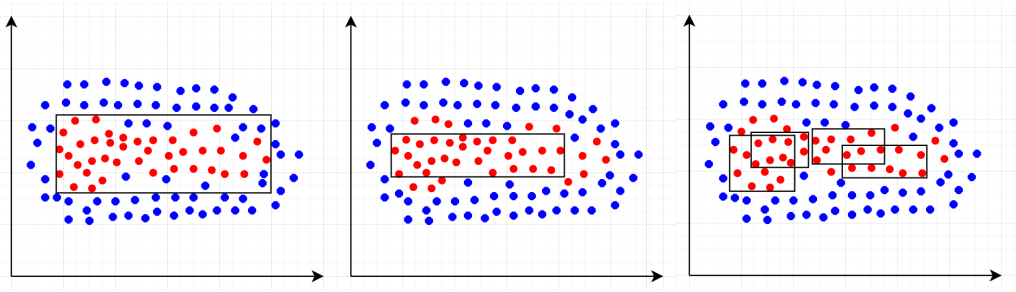


Figure 4.2: *An example of hyper-rectangle drawn around all points of the same class (a), shrunk hyper-rectangle $\mathbb{H}_{sh}$ that is obtained by excluding all points from the opposite class (b) and clustered hyper-rectangles (c) in 2-dimensions. The red dots represent sentences in the embedding space of one class, while the blue dots are embedded sentences that do not belong to that class.*

Formally, for each sentence s in $\mathscr{Y}$ that is not of class c, the algorithm performs the following procedure. If $E(s)$ lies in the current hyper-rectangle $(a_1, b_1), \ldots, (a_m, b_m)$, then for each dimension $j \in [1, \ldots, m]$ we compute the distance whether $E(s)_j$ is closer to a_j or b_j. Without loss of generality, assume a_j is closer. We then compute the number of sentences of class c that would be excluded by replacing a_j with $E(s)_j + \delta$ in the hyper-rectangle where δ is a small positive number (we use e^{-100}). This gives us a penalty for each dimension j, and we exclude s by updating the hyper-rectangle in the dimension that minimises this penalty. The idea is to shrink the hyper-rectangle in the dimensions that exclude as few embedded sentences from the desired class c as possible[3].

[3]Note that this algorithm shrinks exactly one dimension by a minimal amount to exclude the unwanted embedded sentence. This choice keeps the algorithm fast while guaranteeing the subspace to retain the highest number of wanted inputs. However, it is not necessarily the best choice for verification: there might be cases where perturbations of the unwanted input are left inside after shrinking and, if the network classifies them correctly, the subspace can never be verified. For large subspaces, our algorithm might render verification unachievable and more clever algorithms should be explored and discussed.

Exclusion of Unwanted Sentences Via Clustering

An alternative approach to excluding unwanted sentences, is to split the dataset up by clustering semantically similar sentences in the embedding space, and then compute the hyper-rectangles around each cluster individually, as shown in the last graph of Figure 4.2. In this work we will use the k-means algorithm for clustering. We will use the notation $CL(\mathscr{Y}, k)$ to refer to the k-clusters formed by applying it to dataset $\mathscr{Y}$. While in our experiments we have found this is often sufficient to exclude unwanted sentences, it is not guaranteed to do so. Therefore, this method is combined with the shrinking algorithm in our experiments.

Eigenspace Rotation

A final alternative and computationally efficient way of reducing the likelihood that the hyper-rectangles will contain embedded sentences of an unwanted class, is to rotate them to better align to the distribution of the embedded sentences of the desired class in the embedding space. This motivates us to introduce the Eigenspace rotation.

To construct the tightest possible hyper-rectangle, we define a specific method of eigenspace rotation. As shown in Figure 2.1 (C and D), our approach is to calculate a rotation matrix A such that the rotated matrix $\mathscr{X}_{\text{rot}} = \mathscr{X}A$ is better aligned with the axes than $\mathscr{X}$, and therefore $\mathbb{H}(\mathscr{X}_{\text{rot}})$ has a smaller volume. By a slight abuse of terminology, we will refer to $\mathbb{H}(\mathscr{X}_{\text{rot}})$ as the *rotated hyper-rectangle*, even though strictly speaking, we are rotating the data, not the hyper-rectangle itself. In order to calculate the rotation matrix A, we use singular value decomposition [189]. The singular value decomposition of $\mathscr{X}$ is defined as $\mathscr{X} = U\Sigma V^*$, where U is a matrix of left-singular vectors, Σ is a matrix of singular values and V^* is a matrix of right-singular vectors and $\cdot^*$ denotes the conjugate transpose. Intuitively, the right-singular vectors V^* describe the directions in which $\mathscr{X}$ exhibits the most variance. The main idea behind the definition of rotation is to align these directions of maximum variance with the standard canonical basis vectors. Formally, using V^*, we can compute the rotation (or change-of-basis) matrix A that rotates the right-singular vectors onto the canonical standard basis vectors I, where I is the identity matrix. To do this, we observe that $V^*A = I$ implies $V^* = IA^{-1}$, which implies $V^{-1} = A^{-1}$, and thus $V = A$. We thus obtain $\mathscr{X}_{rot} = \mathscr{X}A$ as desired. All hyper-rectangles constructed in this chapter are rotated.

Geometric and Semantic Subspaces

We now apply the abstract definition of a subspace of an embedding space to concrete NLP verification scenarios. Once we know how to define subspaces for a selection of points in the embedding space, the choice remains how to choose those points. The first option is to use ε-cubes around given embedded points, as Example 4 defines. Since this construction does not involve any knowledge about the semantics of sentences, we will call the resulting subspaces *geometric subspaces*. The second choice is to apply semantic perturbations to a sentence in $\mathscr{Y}$, embed the resulting sentences, and then define a subspace around them. We will call the subspaces obtained by this method *semantic perturbation subspaces*, or just *semantic subspaces* for short.

We will finish this section with defining semantic subspaces formally. We will use $\mathscr{P}_t(s)$ to denote an algorithm for generating sentence perturbations of type t, applied to an input sentence s in a random position. In the later sections, we will use t to refer to the different types of perturbations illustrated in Tables 4.1 and 4.2, e.g. character-level insertion, deletion, replacement. Intuitively, given a single sentence we want to generate a set of semantically similar perturbations and then construct a hyper-rectangle around them, as described in Definition 13.

This motivates the following definitions. Given a sentence s, a number b, and a type t, the set $\mathscr{A}_t^b(s) = \{\mathscr{P}_t(s) \mid i \in [1,b]\}$ is the set of b semantic perturbations of type t generated from s. We will use the notation $\mathscr{A}_t^b(\mathscr{Y}) = \bigcup_{s\in\mathscr{Y}} \mathscr{A}_t^b(s)$ to denote the new dataset generated by creating b semantic perturbations of type t around each sentence.

Definition 15 (Semantic Subspace for a Sentence) *Given an embedding function $E : \mathbb{S} \to \mathbb{R}^m$, the semantic subspace for a sentence s is the subspace $\mathbb{H}(\{s\} \cup \mathscr{A}_t^b(s))$. We will refer to a set of such semantic hyper-rectangles over an entire dataset $\mathscr{Y}$ as $\mathbb{H}_t^b(\mathscr{Y}) = \bigcup_{s\in\mathscr{Y}} \mathbb{H}(\{s\} \cup \mathscr{A}_t^b(s))$.*

Example 5 (Construction of Semantic Subspaces) *To illustrate this construction, let us consider the sentence s:* "Can u tell me if you are a chatbot?". *This sentence is one of* 3400 *original sentences of the positive class in the dataset. From this single sentence, we can create six new sentences using the word-level perturbations from Table 4.2 to form $\mathscr{A}_{word}^6(s)$. Once the seven sentences are embedded into the vector space, they form the hyper-rectangle $\mathbb{H}(\{s\} \cup \mathscr{A}_{word}^6(s))$. By repeating this construction for the remaining* 3399 *sentences, we obtain the set of hyper-rectangles $\mathbb{H}_{word}(\mathscr{Y})$ for the dataset.*

Given a sentence s, we embed each sentence in $\mathscr{A}_t^b(s) = \{s_1, \ldots, s_b\}$ into $\mathbb{R}^m$ obtaining vectors $\mathscr{V}_t^b(s) = \{v, v_1, \ldots, v_b\}$ where $v_j = E(s_j)$.

Measuring the Quality of Sentence Embeddings

One of our implicit assumptions in the previous sections, is that the embedding function E maps pairs of semantically similar sentences to nearby points in the embedding space. In Section 4.4.5, we will evaluate the accuracy of this assumption using *cosine similarity*. This metric measures how similar two vectors are in a multi-dimensional space by calculating the cosine of the angle between them:

$$\mathrm{CoS}(v_1, v_2) = \frac{v_1 \cdot v_2}{\|v_1\|\|v_2\|}$$

where $\cdot$ is the dot product and $\|v\| = \sqrt{v \cdot v}$. The resulting value ranges from 0 to 1. A value of 1 indicates that the vectors are parallel (highest similarity), while 0 means that the vectors are orthogonal (no similarity).

4.2.5 Training

As introduced in Chapter 2, robust training is crucial for improving a model's ability to maintain correct predictions under perturbations. In the NLP setting, robust training must account for the discrete, symbolic nature of language as well as the structure of embedding space.

This section adapts two standard robustness training techniques — data augmentation and adversarial training — to the NLP domain, with modifications that respect the structure of embedding subspaces defined in Section 4.2.4. Our goal is to compare the effectiveness of these methods in producing models that are both verifiable and generalisable.

Data Augmentation. In this approach, semantic perturbations are generated at the character, word, and sentence levels prior to training. These perturbations are added to the dataset, and the network is trained on the augmented set using standard stochastic gradient descent. This method improves robustness by exposing the model to known linguistic variability during training.

Adversarial Training. We adapt the PGD method [9], introduced in Section 2.1.3, for NLP by applying it in embedding space. Specifically, we modify PGD to operate over custom-defined hyper-rectangles rather than ε-balls. The key difference lies in the step size: in the standard algorithm, the step size $\gamma \in \mathbb{R}$ is a scalar, applying the same update across all dimensions. However, since the width of our hyper-rectangles may vary substantially across dimensions, we generalise γ to a vector in $\mathbb{R}^m$, allowing per-dimension scaling. The dot $\cdot$ operation becomes element-wise multiplication. This customised PGD training seeks adversarial points within each subspace that maximally increase the loss, and updates the model to classify them correctly — improving robustness to perturbations specifically tailored to the structure of the embedding space.

4.2.6 Choice of Verification Algorithm

As stated earlier, our approach in this study involves the utilization of cutting-edge tools for DNN verification. Initially, we employ ERAN [190], a state-of-the-art abstract interpretation-based method. This choice is made over IBP due to its ability to yield tighter bounds. Subsequently, we conduct comparisons and integrate Marabou [33], a state-of-the-art complete verifier. This enables us to attain the highest verification percentage, maximizing the tightness of the bounds. Additionally, we incorporate $\alpha\beta$-CROWN[35, 36], the best-performing verifier in the VNN-COMP 2024 competition, known for its efficiency in linear bound propagation and branch-and-bound techniques. Moreover, we utilise CORA[31], an abstract interpretation-based verifier that supports zonotope-based verification, allowing us to compare hyper-rectangles and zonotopes in our verification experiments. We will use notation $\mathscr{V}$ to refer to a verifier abstractly.

While Vehicle is not used for verification in this chapter — primarily due to its limited back-end support (currently only Marabou) — we still adopt it to illustrate how our robustness constraints could be encoded in a high-level, symbolic language. This helps conceptually bridge to the NIDS case in Chapter 5, where such symbolic expressivity is essential. Figure 4.3 provides a Vehicle encoding of an NLP robustness constraint based on a hyper-rectangle in the embedding space. Note how lines 19–20 define the bounds of the hyper-rectangle. In contrast, Chapter 5 constructs these bounds programmatically and passes them as parameters to Vehicle. Nonetheless, this example provides a simplified, self-contained illustration of the same geometric principle.

```
-- Illustrative Vehicle spec for NLP

@network
classifier : Tensor Rat [30] -> Vector Rat 2

advises : Tensor Rat [30] -> Index 2 -> Bool
advises x i = forall j . j != i => classifier x ! i > classifier x ! j

@parameter(infer=True)
n : Nat

@dataset
inputs : Tensor Rat [n, 30]

-- Build geometric bounds over the dataset
vectorMin : Tensor Rat [30]
vectorMax : Tensor Rat [30]

hyperRectangle : Tensor Rat [30] -> Bool
hyperRectangle x = forall i . vectorMin ! i <= x ! i <= vectorMax ! i

@property
property : Bool
property = forall x . hyperRectangle x => advises x 0
```

Figure 4.3: *An illustrative Vehicle specification for NLP robustness. While not used for verification here, it formalises the geometric subspace constraint described in Section 4.2.4.*

4.3 Characterisation of Verifiable Subspaces

In this Section, we provide key results in support of **Contribution 1** formulated in the introduction:

- We start with introducing the metric of *generalisability* of (verified) subspaces and set-up some baseline experiments.
- We introduce the problem of the *verifiability-generalisability trade-off* in the context of geometric subspaces.
- We show that, compared to geometric subspaces, the use of semantic subspaces helps to find a better balance between generalisability and verifiability.
- Finally, we show that adversarial training based on semantic subspaces results in DNNs that are both more verifiable and more generalisable than those obtained with other forms of robust training.

4.3.1 Metrics for Understanding the Properties of Embedding Spaces

Let us start with recalling the existing standard metrics used in DNN verification. Recall that we are given an NLP dataset $\mathscr{Y} = \{s_1, \ldots, s_q\}$, moreover we assume that each s_i is assigned a *correct class* from $C = \{c_1, \ldots, c_n\}$. We restrict to the case of binary classification in this thesis for simplicity, so we will assume $C = \{c_1, c_2\}$. Furthermore, we are given an embedding function $E : \mathbb{S} \to \mathbb{R}^m$, and a network $N : \mathbb{R}^m \to \mathbb{R}^n$. Usually n corresponds to the number of classes, and thus in case of binary classification, we have $N : \mathbb{R}^m \to \mathbb{R}^2$. An embedded sentence $s \in \mathscr{Y}$ is *classified* as class c if the value of c in $N(E(s))$ is higher than all other classes.

Accuracy. The most popular metric for measuring the performance of the network is the *accuracy* of N, which is measured as a percentage of sentences in $\mathscr{Y}$ that are assigned to a correct class by N. Note that this metric only checks a finite number of points in $\mathbb{R}^m$ given by the dataset.

Verifiability. A verifier $\mathscr{V}$ takes a network N, a subspace $\mathscr{S}$ and its designated class c as an input, and outputs 1 if it can prove that N assigns all points in the subspace $\mathscr{S}$ to the class c and 0 otherwise. Consider a verification problem with multiple subspaces $\{\mathscr{S}_1, \ldots, \mathscr{S}_l\}$, where all the points in each subspace should be assigned to a specific class $c_i \in \{c_1, \ldots, c_n\}$. In the literature, the most popular metric to measure success rate of the given verifier on $\{\mathscr{S}_1, \ldots, \mathscr{S}_l\}$ is *verifiability*:

Definition 16 (Verifiability) *Given a set of subspaces $\mathscr{S}_1, \ldots, \mathscr{S}_l$ each assigned to classes $c_1, \ldots, c_l$, then the verifiability is the percentage of such subspaces successfully verified:*

$$\mathscr{W}(\mathscr{S}_1, \ldots, \mathscr{S}_l, c_1, \ldots, c_l) = \frac{\sum_{i=0}^{l} \mathscr{V}(N, \mathscr{S}_i, c_i)}{l}$$

Relation to CSat (Definition 10). CSat evaluates, for each test input $\hat{x}$, whether the target property holds throughout its ε-ball $\mathbb{B}(\hat{x}, \varepsilon)$; it is therefore tied to *local* regions of the ε-ball form. By contrast, Definition 16 treats the *region* as the unit of analysis and is agnostic to the region shape. If we take the regions in Definition 16 to be the ε-balls around the same anchor inputs used for CSat, then the verifiability score equals the CSat value on that anchor set. When the norm is ℓ_∞, ε-balls coincide with ε-cubes, and Definition 16 then *strictly generalises* CSat by also accommodating arbitrary axis-aligned hyper-rectangles (global regions).

However, suppose we have a subspace $\mathscr{S}$ that verifiably consists only of vectors

that are assigned to a class c by N. Because of the embedding gap, it is difficult to calculate how many valid unseen sentences outside of $\mathscr{Y}$ will be mapped into $\mathscr{S}$ by E, and therefore how much utility there is in verifying $\mathscr{S}$. In an extreme case it is possible to have 100% verifiability and yet the verified subspaces will not contain any unseen sentences.

Generalisability. Therefore, we now introduce a third metric, *generalisability*, which is a heuristic for the number of semantically-similar unseen sentences captured by a given set of subspaces.

Definition 17 (Generalisability) *Given a set of subspaces $\mathscr{S}_1, \ldots, \mathscr{S}_l$ and a target set of embeddings V the* generalisability *of the subspaces is measured as the percentage of the embedded vectors that lie in the subspaces:*

$$\mathscr{G}(V, \mathscr{S}_1, \ldots, \mathscr{S}_l) = \frac{|V \cap \bigcup_{i=1}^{l} \mathscr{S}_i|}{|V|}$$

In this work we will generate the target set of embeddings V as $\bigcup_{s \in \mathscr{Y}} \mathscr{V}_t^b(s)$ where $\mathscr{Y}$ is a dataset, t is the type of semantic perturbation, b is the number of perturbations and $\mathscr{V}_t^b(s)$ is the embeddings of the set of semantic perturbations $\mathscr{A}_t^b(s)$ around s generated using $\mathscr{P}_t$, as described in Section 4.2.4.

Note that $\mathscr{P}_t$ can be given by a collection of different perturbation algorithms and their kinds. The key assumption is that $\mathscr{A}_t^b(s)$ contains valid sentences semantically similar to s and belonging to the same class. Assuming that membership of $\mathscr{S}$ is easy to compute, then this metric is also easy to compute as the set $\mathscr{A}_t^b(s)$ is finite and of size b, and therefore so is $\mathscr{V}_t^b(s)$. Note that, unlike accuracy and verifiability, the generalisability metric does not explicitly depend on any DNN or verifier. However, in this work we only study generalisability of verifiable subspaces, and thus the existence of a verified network N will be assumed. Furthermore, the verified subspaces we study in this work will be constructed from the dataset via the methodology described in Definition 15.

4.3.2 Baseline Experiments for Understanding the Properties of Embedding Spaces

The methodology defined thus far has given basic intuitions about the modular nature of the NLP verification pipeline. Bearing this in mind, it is important to start our analysis with the general study of basic properties of the embedding subspaces, which is our main interest in this work, and suitable baselines.

Benchmark datasets will be abbreviated as "*RUAR*" and "*Medical*". We use $\mathscr{D}^{pos}$ to refer to the set of sentences in the training dataset with a positive class (i.e. a question asking the identity of the model, and a medical query respectively), and $\mathscr{D}^{neg}$ to refer to the remaining sentences. For a benchmark network $N : \mathbb{R}^m \to \mathbb{R}^2$, we train a medium-sized fully-connected DNN (with 2 layers of size (128, 2) and input size 30) using *stochastic gradient descent* and *cross-entropy loss*. The main requirement for a benchmark network is its sufficient accuracy, see Table 4.3.

Model	**Adversarial training**	**Train Accuracy RUAR**	**Test Accuracy RUAR**	**Train Accuracy Medical**	**Test Accuracy Medical**
N_{base}	No	93.87 ± 0.14%	93.57 ± 0.18%	**96.32 ± 0.05**%	94.49 ± 0.26%

Table 4.3: *Mean and standard deviation of the accuracy of the baseline DNN on the RUAR and the Medical datasets. All experiments are replicated five times.*

For the choice of benchmark subspaces, we use the following two extreme sets of geometric subspaces:

1. the singleton set containing the maximal subspace $SH(\mathbb{H}(\mathscr{D}^{pos}))$ around all embedded sentences of the positive class *pos* in $\mathscr{Y}$. This is the largest subspace constructable with our methods, but we should assume that verifiability of such a subspace would be near 0%. It is illustrated in the first graph of Figure 4.2.

2. the set of minimal subspaces $\mathbb{H}(\mathscr{D}^{pos}, 0.005)$ given by ε-cubes around each embedded sentence of class *pos* in $\mathscr{Y}$, where $\varepsilon = 0.005$ is chosen to be sufficiently small to give very high verifiability. This is illustrated in the first graph of Figure 2.1.

We first seek to understand the geometric properties (e.g. volume, ε values) and verifiability figures for these two extremes.

4.3.3 Verifiability-Generalisability Trade-off for Geometric Subspaces

The number and average volume of the hyper-rectangles that will make up our verified subspaces are shown in Table 4.4. Generally, we use the following naming convention for our experiments: $\mathbb{H}_m$ denotes a hyper-rectangle obtained using a method m. For example, RUAR dataset contains 3400 sentences of the positive class, and therefore the experiment $\mathbb{H}_{\varepsilon=0.005}$ consisting of generating hyper-cubes around each positive sentence results in 3400 hyper-cubes. Using clustering, we obtain a set of 50, 100, 200, 250 clusters denoted as $\mathbb{H}_{50} - \mathbb{H}_{250}$ and using the shrinking algorithm we obtain $\mathbb{H}_{sh}$.

Notice the consistent reduction of volume in Table 4.4, from $\mathbb{H}_{sh}$ to $\mathbb{H}_{50}$ - $\mathbb{H}_{250}$ and ultimately to $\mathbb{H}_{\varepsilon=0.005}$. There are several orders of magnitude between the largest and the smallest subspace.

Experiment name	Hyper-rectangles construction method	Avg. volume of hyper-rectangles RUAR	Number of hyper-rectangles RUAR	Avg. volume of hyper-rectangles Medical	Number of hyper-rectangles Medical
$\mathbb{H}_{sh}$	Hyper-rectangle around the entire dataset shrunk to exclude all negative examples - $SH(\mathbb{H}(\mathscr{D}^{pos}), \mathscr{Y}, c^{pos})$	7.55e-11	1	2.60e-09	1
$\mathbb{H}_{50}$	Set of hyper-rectangles on the dataset separated into 50 clusters - $\mathbb{H}(CL(\mathscr{D}^{pos}, 50))$	1.02e-16	50	6.56e-15	50
$\mathbb{H}_{100}$	Set of hyper-rectangles on the dataset separated into 100 clusters - $\mathbb{H}(CL(\mathscr{D}^{pos}, 100))$	6.23e-18	100	3.25e-17	100
$\mathbb{H}_{200}$	Set of hyper-rectangles on the dataset separated into 200 clusters - $\mathbb{H}(CL(\mathscr{D}^{pos}, 200))$	3.31e-20	200	4.67e-19	200
$\mathbb{H}_{250}$	Set of hyper-rectangles on the dataset separated into 250 clusters - $\mathbb{H}(CL(\mathscr{D}^{pos}, 250))$	6.42e-22	250	2.42e-20	250
$\mathbb{H}_{\varepsilon=0.05}$	Set of ε-cubes around all positive sentences in the dataset - $\mathbb{H}(\mathscr{D}^{pos}, 0.05)$	1.00e-30	3400	1.00e-30	989
$\mathbb{H}_{\varepsilon=0.005}$	Set of ε-cubes around all positive sentences in the dataset - $\mathbb{H}(\mathscr{D}^{pos}, 0.005)$	1.00e-60	3400	1.00e-60	989

Table 4.4: *Sets of geometric subspaces used in the experiments, their cardinality and average volumes of hyper-rectangles. All shapes are eigenspace rotated for better precision.*

Verifiability of Geometric Subspaces

Next, we pass each set of hyper-rectangles and the given network to the ERAN verifier and measure verifiability. Table 4.5 shows that, as expected, the shrunk hyper-rectangle $\mathbb{H}_{sh}$ achieves 0% verifiability, and the various clustered hyper-rectangles ($\mathbb{H}_{50}$, $\mathbb{H}_{100}$ $\mathbb{H}_{200}$, $\mathbb{H}_{250}$) achieve at most negligible verifiability. In contrast, the baseline $\mathbb{H}_{\varepsilon=0.005}$ achieves up to 99.60% verifiability. This suggests that $\varepsilon = 0.005$ is a good benchmark for a different extreme. Table 4.4 can give us an intuition of why $\mathbb{H}_{\varepsilon=0.005}$ has notably higher verifiability than the other hyper-rectangles: the volume of $\mathbb{H}_{\varepsilon=0.005}$ is several orders of magnitude smaller.. We call this effect **low verifiability of the high-volume subspaces**.

Dataset	Model	$\mathbb{H}_{sh}$	$\mathbb{H}_{50}$	$\mathbb{H}_{100}$	$\mathbb{H}_{200}$	$\mathbb{H}_{250}$	$\mathbb{H}_{\varepsilon=0.05}$	$\mathbb{H}_{\varepsilon=0.005}$
RUAR	N_{base}	0.00%	0.00%	1.33%	**0.52**%	0.41%	0.00%	**88.67**%
Medical	N_{base}	0.00%	0.00%	0.00%	2.10%	**4.08**%	5.00%	**97.86**%

Table 4.5: *Verifiability of the baseline DNN on the RUAR and the Medical datasets, for a selection of geometric subspaces; using the ERAN verifier.*

Tables 4.4 and 4.5 suggest that smaller subspaces are more verifiable. One may also conjecture that they are less generalisable (as they will contain fewer embedded sentences). We now will confirm this via experiments; we are particularly interested in

understanding how quickly generalisability deteriorates as verifiability increases.

Generalisability of Geometric Subspaces

To test generalisability, we algorithmically generate a new dataset $\mathscr{A}_t^b(\mathscr{D}^{pos})$ containing its semantic perturbations, using the method described in Section 4.2.2. The choice to use only positive sentences is motivated by the nature of the chosen datasets - both Medical and RUAR sentences split into:

- a positive class, that contains sentences with one intended semantic meaning (they are medical queries, or they are questions about robot identity); and
- a negative class that represents "all other sentences". These "other sentences" are not grouped by any specific semantic meaning and therefore do not form one coherent semantic category.

However Section 4.4 will make use of $\mathscr{A}_t^b(\mathscr{D}^{neg})$ in the context of discussing the embedding error of verified subspaces.

For the perturbation type t, in this experiment we take a combination of the different perturbations algorithms[4] described in Section 4.2.2. Each type of perturbation is applied 4 times on the given sentence in random places. The resulting datasets of semantically perturbed sentences are therefore approximately two orders of magnitude larger than the original datasets (see Table 4.6), and contain unseen sentences of similar semantic meaning to the ones present in the original datasets *RUAR* and *Medical*.

Dataset	Experiment	Avg. Volume of hyper-rectangles	Generalisability (%)	Number of sentences contributing to generalisability	Total Sentences in $\mathscr{A}_t^b(\mathscr{D}^{pos})$
RUAR	$\mathbb{H}_{\varepsilon=0.005}$	1.00e-60	1.95	2821	144500
	$\mathbb{H}_{\varepsilon=0.05}$	1.00e-30	38.47	55592	144500
	$\mathbb{H}_{sh}$	7.55e-11	50.91	73561	144500
Medical	$\mathbb{H}_{\varepsilon=0.005}$	1.00e-60	0.09	10	11209
	$\mathbb{H}_{\varepsilon=0.05}$	1.00e-30	28.49	3194	11209
	$\mathbb{H}_{sh}$	2.6e-09	37.13	4162	11209

Table 4.6: *Generalisability of the selected geometric subspaces* $\mathbb{H}_{\varepsilon=0.005}$, $\mathbb{H}_{\varepsilon=0.05}$ *and* $\mathbb{H}_{sh}$, *measured on the sets of semantic perturbations* $\mathscr{A}^b_{t_{RAUR}}(\mathscr{D}^{pos})$ *and* $\mathscr{A}^b_{t_{medical}}(\mathscr{D}^{pos})$.

[4]For RUAR, $t_{RUAR} = \{$ *character insertion, character deletion, character replacement, character swapping, character repetition, word deletion, word repetition, word negation, word singular/plural verbs, word order, word tense* $\}$. For the Medical dataset, $t_{Medical} = \{$ *character insertion, character deletion, character replacement, character swapping, character repetition, word deletion, word repetition, word negation, word singular/plural verbs, word order, word tense, sentence polyjuice*$\}$.

Table 4.6 shows that the **most verifiable** subspace $\mathbb{H}_{\varepsilon=0.005}$ is the **least generalisable**. This means $\mathbb{H}_{\varepsilon=0.005}$ may not contain any valid new sentences apart from the one for which it was formed! At the same time, $\mathbb{H}_{\varepsilon=0.05}$ has up to 48% of generalisability at the expense of only up to 5% of verifiability (cf. Table 4.5). The effect of the generalisability vs verifiability trade-off can thus be rather severe for geometric subspaces.

This experiment demonstrates the importance of using the generalisability metric: if one only took into account the verifiability of the subspaces one would choose $\mathbb{H}_{\varepsilon=0.005}$, obtaining *mathematically sound but pragmatically useless results*. We argue that this is a strong argument for including generalisability as a standard metric in reporting NLP verification results in the future.

4.3.4 Verifiability-Generalisability Trade-off for Semantic Subspaces

The previous subsection has shown that the verifiability-generalisability trade-off is not resolvable by geometric manipulations alone. In this section we argue that using semantic subspaces can help to improve the effects of the trade-off. The main hypothesis that we are testing is: *semantic subspaces constructed using semantic-preserving perturbations are more precise, and this in turn improves both verifiability and generalisability*.

We will use the construction given in Definition 15. As Table 4.7 illustrates, we construct several semantic hyper-rectangles on sentences of the positive class using *character-level* ($\mathbb{H}_{char}$, $\mathbb{H}_{del.}$, $\mathbb{H}_{ins.}$, $\mathbb{H}_{rep.}$, $\mathbb{H}_{repl.}$, $\mathbb{H}_{swap.}$), word-level ($\mathbb{H}_{word}$) and sentence-level perturbations ($\mathbb{H}_{pj}$). The subscripts $_{char}$ and $_{word}$ refer to the *kind* of perturbation algorithm, while $_{del.}$, $_{ins.}$, $_{rep.}$, $_{repl.}$, $_{swap.}$ and $_{pj}$ refer to the *type* of perturbation, where pj stands for Polyjuice (see Section 4.2.2). Notice comparable volumes of all these shapes, and compare with $\mathbb{H}_{\varepsilon=0.05}$.

Verifiability of Semantic Subspaces

We pass each set of hyper-rectangles and the network N_{base} to the verifiers ERAN and Marabou to measure verifiability of the subspaces. Table 4.8 illustrates the verification results obtained using ERAN. From the table, we can infer that the verifiability of our semantic hyper-rectangles is indeed higher than that of the geometrically-defined hyper-rectangles (Table 4.5). Furthermore, our semantic hyper-rectangles, while unable to reach the verifiability of $\mathbb{H}_{\varepsilon=0.005}$, achieve notable higher verification than its counterpart of comparable volume $\mathbb{H}_{\varepsilon=0.05}$. From this experiment, we conclude that not only **volume**,

Experiment name	Hyper-rectangles construction method	Avg. volume of hyper-rectangles RUAR	Number of hyper-rectangles RUAR	Avg. volume of hyper-rectangles Medical	Number of hyper-rectangles Medical
$\mathbb{H}_{char}$	Set of hyper-rectangles for character perturbations	1.54e-30	3400	7.66e-31	989
$\mathbb{H}_{word}$	Set of hyper-rectangles for word perturbations	1.28e-30	3400	-	-
$\mathbb{H}_{pj}$	Set of hyper-rectangles for polyjuice sentence perturbations	-	-	2.01e-28	989
$\mathbb{H}_{swap.}$	Set of hyper-rectangles for swapping perturbations	1.57e-31	3400	3.42e-31	989
$\mathbb{H}_{repl.}$	Set of hyper-rectangles for replacement perturbations	9.84e-31	3400	3.43e-31	989
$\mathbb{H}_{del.}$	Set of hyper-rectangles for deletion perturbations	3.46e-31	3400	1.24e-32	989
$\mathbb{H}_{ins.}$	Set of hyper-rectangles for insertion perturbations	3.21e-31	3400	9.11e-33	989
$\mathbb{H}_{rep.}$	Set of hyper-rectangles for repetition perturbations	1.56e-31	3400	1.06e-32	989

Table 4.7: *Sets of semantic subspaces used in the experiments, their cardinality and average volumes of hyper-rectangles. All shapes are eigenspace rotated for better precision.*

but also **precision** of the subspaces has an impact on their **verifiability**.

Dataset	Model	$\mathbb{H}_{\varepsilon=0.05}$	$\mathbb{H}_{word}$	$\mathbb{H}_{char}$	$\mathbb{H}_{del.}$	$\mathbb{H}_{ins.}$	$\mathbb{H}_{rep.}$	$\mathbb{H}_{repl.}$	$\mathbb{H}_{swap.}$	$\mathbb{H}_{pj}$
RUAR	N_{base}	0.00%	1.80%	0.87%	1.62%	2.63%	1.66%	0.94%	2.07%	-
Medical	N_{base}	5.00%	-	39.71%	39.62%	44.66%	48.71%	37.49%	42.60%	**50.09%**

Table 4.8: *Verifiability of the baseline DNN on the RUAR and the Medical datasets, for a selection of semantic subspaces; using the ERAN verifier.*

Following these results, Table 4.9 reports the verification results using Marabou instead of ERAN. As shown, Marabou is able to verify up to 66.83% ($\mathbb{H}_{rep.}$), while ERAN achieves at most 50.09%. This shows that Marabou outperforms ERAN. This is most likely due to the fact that Reluplex algorithm of Marabou achieves better precision on ReLU networks, that we use. Overall, the Marabou experiment confirms the trends of improved verifiability shown by ERAN and thus confirms our hypothesis about importance of shape precision.

Dataset	Model	$\mathbb{H}_{\varepsilon=0.05}$	$\mathbb{H}_{word}$	$\mathbb{H}_{char}$	$\mathbb{H}_{del.}$	$\mathbb{H}_{ins.}$	$\mathbb{H}_{rep.}$	$\mathbb{H}_{repl.}$	$\mathbb{H}_{swap.}$	$\mathbb{H}_{pj}$
RUAR	N_{base}	1.79%	11.69%	4.88%	4.35%	9.72%	9.46%	5.65%	8.07%	-
Medical	N_{base}	37.96%	-	64.03%	64.15%	64.65%	**66.83%**	64.75%	64.36%	61.57%

Table 4.9: *Verifiability of the baseline DNN on the RUAR and the Medical datasets, for a selection of semantic subspaces; using the Marabou verifier.*

Generalisability of Semantic Subspaces

It remains to establish whether the more verifiable semantic subspaces are also more generalisable. Whereas Table 4.6 compared the generalisability of $\mathbb{H}_{\varepsilon=0.005}$ and $\mathbb{H}_{\varepsilon=0.05}$

with that of $\mathbb{H}_{sh}$, Table 4.10 compares their generalisability to the most verifiable semantic subspaces, $\mathbb{H}_{word}$ and $\mathbb{H}_{pj}$. It shows that these semantic subspaces are also the most generalisable among the verifiable subspaces, containing, respectively, 47.67% and 28.74% of the unseen sentences. Note that among all the experiments, only $\mathbb{H}_{sh}$ has higher generalisability, but its verifiability is 0.

Dataset	Experiment	Avg. Volume of hyper-rectangles	Generalisability (%)	Number of sentences contributing to generalisability	Total Sentences in $\mathscr{A}^b_t(\mathscr{D}^{pos})$
RUAR	$\mathbb{H}_{\varepsilon=0.005}$	1.00e-60	1.95	2821	144500
	$\mathbb{H}_{\varepsilon=0.05}$	1.00e-30	38.47	55592	144500
	$\mathbb{H}_{word}$	1.28e-30	**47.67**	**68882**	144500
Medical	$\mathbb{H}_{\varepsilon=0.005}$	1.00e-60	0.09	10	11209
	$\mathbb{H}_{\varepsilon=0.05}$	1.00e-30	28.49	3194	11209
	$\mathbb{H}_{pj}$	2.01e-28	**28.74**	**3222**	11209

Table 4.10: *Generalisability of the selected geometric subspaces $\mathbb{H}_{\varepsilon=0.005}$ and $\mathbb{H}_{\varepsilon=0.05}$ and the semantic subspaces $\mathbb{H}_{word}$ and $\mathbb{H}_{pj}$, measured on the sets of semantic perturbations $\mathscr{A}^b_{t_{RUAR}}(\mathscr{D}^{pos})$ and $\mathscr{A}^b_{t_{medical}}(\mathscr{D}^{pos})$. Note that the generalisability of $\mathbb{H}_{sh}$ (Table 4.6), despite it having the volume 19 order of magnitudes bigger, is only 3% greater than $\mathbb{H}_{word}$.*

We thus infer that using semantic subspaces is effective for bridging the verifiability-generalisability gap, with precise subspaces performing somewhat better than ε-cubes of the same volume; however both beating the smallest ε-cubes from Section 4.3.2 of comparable verifiability. Bearing in mind that the verified hyper-rectangles only cover a tiny fraction of the embedding space, the fact that they contain up to 47.67% of randomly generated new sentences is an encouraging result, the likes of which have not been reported before. To substantiate this claim, we define the training embedding space as the hyper-rectangle that encloses all sentences on the dataset. We show in Table 4.11 the percentage of the 'training embedding space' covered by our best hyper-rectangles $\mathbb{H}_{\varepsilon=0.005}$, $\mathbb{H}_{\varepsilon=0.05}$, $\mathbb{H}_{word}$ and $\mathbb{H}_{pj}$.

Dataset	Experiment	Total Volume of Hyper-rectangles	Training Embedding Space Covered (%)
RUAR	$\mathbb{H}_{\varepsilon=0.005}$	2.89e-57	4.71e-53
	$\mathbb{H}_{\varepsilon=0.05}$	2.89e-27	4.71e-23
	$\mathbb{H}_{word}$	3.7e-27	6.03e-23
Medical	$\mathbb{H}_{\varepsilon=0.005}$	9.89e-58	6.92e-53
	$\mathbb{H}_{\varepsilon=0.05}$	9.89e-28	6.92e-23
	$\mathbb{H}_{pj}$	1.63e-25	1.14e-20

Table 4.11: *Total volume and percentage of the training embedding space covered by our best hyper-rectangles. The total volume of the training embedding space for RUAR is $6.14e-5$, and for Medical is $1.43e-5$.*

4.3.5 Adversarial Training on Semantic Subspaces

In this section, we study the effects that adversarial training methods have on the verifiability of the previously defined subspaces in Tables 4.4 and 4.7. By comparing the effectiveness of the different training approaches described in Section 4.2.5, we show in this section that *adversarial training based on our new semantic subspaces is the most efficient*. Three kinds of training are deployed in this section:

1. *No robustness training*. The baseline network is N_{base} from the previous experiments, which has not undergone any robustness training.

2. *Data augmentation*. We obtain three augmented datasets $\mathscr{Y} \cup \mathscr{A}^5_{char}(\mathscr{D}^{pos})$, $\mathscr{Y} \cup \mathscr{A}^6_{word}(\mathscr{D}^{pos})$ and $\mathscr{A}^5_{pj}(\mathscr{D}^{pos})$ where $\mathscr{A}(\cdot)$ is defined in Section 4.3.4. The subscripts *char* and *word* denote the type of perturbation as detailed in Tables 4.1 and 4.2, while the subscript *pj* refers to the sentence level perturbations generated with Polyjuice. We train the baseline architecture, using the standard stochastic gradient descent and cross entropy loss, on the augmented datasets, and obtain DNNs $N_{char-aug}$, $N_{word-aug}$ and N_{pj-aug}.

3. *PGD adversarial training with geometric and semantic hyper-rectangles.* Instead of using the standard ε-cube as the PGD subspace $\mathscr{S}$, we use the various hyper-rectangles defined in Tables 4.4 & 4.7. We refer to a network trained with the PGD algorithm on the hyper-rectangle associated with experiment $\mathbb{H}_{name}$ as $N_{name-adv}$. For example, for the previous experiment $\mathbb{H}_{sh}$, we obtain the network N_{sh-adv} by adversarially training the benchmark architecture on the associated subspace $\mathscr{S} = SH(\mathbb{H}(\mathscr{D}^{pos}), \mathscr{Y}, c^{pos})$.

See Tables 4.12 & 4.15 for full listing of the networks we obtain in this way. We call DNNs of second and third type *robustly trained networks*. We keep the geometric and semantic subspaces from the previous experiments (shown in Table 4.7) to compare how training affects their verifiability.

Following the same evaluation methodology of experiments as in Sections 4.3.2 and 4.3.4, we use the verifiers ERAN and Marabou to measure verifiability of the subspaces. Table 4.12 reports accuracy of the robustly trained networks, while the verification results are presented in Tables 4.13 and 4.14. From Table 4.12 we can see that networks trained with data augmentation achieve similar nominal accuracy to networks trained with adversarial training. However, the most prominent difference is exposed in Tables 4.13

and 4.14: **adversarial training** effectively **improves the verifiability** of the networks, while data augmentation actually decreases it.

Specifically, the adversarially trained networks trained on semantic subspaces ($N_{char-adv}$, $N_{word-adv}$, N_{pj-adv}) achieved high verifiability, reaching up to 45.87% for RUAR and up to 83.48% for the Medical dataset. This constitutes a significant improvement of the *verifiability* results compared to N_{base}. Looking at nuances, there does not seem to be a single winner subspace when it comes to adversarial training, and indeed in some cases $\mathbb{H}_{\varepsilon=0.05}$ wins over more precise subspaces. All of the subspaces in Table 4.7 have very similar volume, which accounts for improved performance across all experiments. The particular peaks in performance then come down to particularities of a specific semantic attack that was used while training. For example, the best performing networks are those trained with Polyjuice attack, the strongest form of attack in our range. Thus, if the kind of attack is known in advance, the **precision of hyper-rectangles** can be further tuned.

Model	Dataset	Train Accuracy RUAR	Test Accuracy RUAR	Train Accuracy Medical	Test Accuracy Medical
$N_{char-aug}$	$\mathscr{D} \cup \mathscr{A}^5_{char}(\mathscr{D}^{pos})$	95.62 ± 0.26%	93.20 ± 0.35%	99.08 ± 0.06%	93.46 ± 0.30%
$N_{word-aug}$	$\mathscr{D} \cup \mathscr{A}^6_{word}(\mathscr{D}^{pos})$	98.57 ± 0.06%	94.59 ± 0.36%	-	-
N_{pj-aug}	$\mathscr{D} \cup \mathscr{A}^5_{pj}(\mathscr{D}^{pos})$	-	-	98.19 ± 0.09%	93.19 ± 0.39%
$N_{char-adv}$	$\mathscr{Y}$	93.26 ± 0.19%	92.51 ± 0.38%	96.27 ± 0.05%	95.09 ± 0.16%
$N_{word-adv}$	$\mathscr{Y}$	93.68 ± 0.16%	92.37 ± 0.29%	-	-
N_{pj-adv}	$\mathscr{Y}$	-	-	95.05 ± 0.19%	93.49 ± 0.32%
$N_{\varepsilon=0.05-adv}$	$\mathscr{Y}$	94.01 ± 0.17%	92.24 ± 0.19%	96.05 ± 0.09%	95.04 ± 0.24%

Table 4.12: *Accuracy of the robustly trained DNNs on the RUAR and the Medical datasets. $\mathscr{Y}$ stands for either RUAR or Medical depending on the column.*

Dataset	Model	$\mathbb{H}_{\varepsilon=0.05}$	$\mathbb{H}_{word}$	$\mathbb{H}_{char}$	$\mathbb{H}_{del.}$	$\mathbb{H}_{ins.}$	$\mathbb{H}_{rep.}$	$\mathbb{H}_{repl.}$	$\mathbb{H}_{swap.}$	$\mathbb{H}_{pj}$
RUAR	$N_{char-aug}$	0.00%	0.24%	0.00%	0.51%	1.38%	1.09%	0.35%	1.06%	-
	$N_{word-aug}$	0.00%	0.24%	0.00%	0.42%	0.31%	0.57%	0.25%	0.92%	-
	$N_{char-adv}$	0.00%	8.97%	**4.43%**	**4.81%**	**9.86%**	**11.3%**	**6.91%**	**8.51%**	-
	$N_{word-adv}$	0.04%	**10.75%**	4.05%	4.36%	8.60%	9.52%	6.81%	7.45%	-
	$N_{\varepsilon=0.05-adv}$	**0.12%**	10.16%	4.18%	4.04%	8.91%	10.17%	6.52%	7.36%	-
Medical	$N_{char-aug}$	0.00%	-	7.59%	5.28%	12.84%	11.05%	7.92%	7.40%	26.97%
	N_{pj-aug}	0.00%	-	10.31%	8.49%	15.67%	14.90%	9.18%	10.58%	28.59%
	$N_{char-adv}$	5.28%	-	50.12%	49.78%	53.99%	57.76%	48.02%	52.07%	55.44%
	N_{pj-adv}	2.83%	-	47.11%	46.14%	52.12%	56.14%	44.59%	48.27%	57.36%
	$N_{\varepsilon=0.05-adv}$	**8.68%**	-	**51.60%**	**50.31%**	**55.67%**	**58.52%**	**50.10%**	**53.65%**	**59.76%**

Table 4.13: *Verifiability of the robustly trained DNNs on the RUAR and the Medical datasets, for a selection of semantic subspaces; using the ERAN verifier.*

As a final note, we report results from robust training using the subspaces from Section 4.3.2 in Table 4.4. Table 4.15 reports the accuracy and the details of the robustly trained networks on those subspaces, while the verification results are presented in Table 4.16. These tables further demonstrate the importance of volume, and show that **subspaces that are too big still achieve negligible verifiability even after adversarial**

Dataset	Model	$\mathbb{H}_{\varepsilon=0.05}$	$\mathbb{H}_{word}$	$\mathbb{H}_{char}$	$\mathbb{H}_{del.}$	$\mathbb{H}_{ins.}$	$\mathbb{H}_{rep.}$	$\mathbb{H}_{repl.}$	$\mathbb{H}_{swap.}$	$\mathbb{H}_{pj}$
RUAR	$N_{char-aug}$	0.72%	13.90%	8.49%	7.92%	13.67%	15.50%	9.56%	11.88%	-
	$N_{word-aug}$	0.24%	11.30%	3.87%	4.05%	8.27%	8.84%	5.71%	7.72%	-
	$N_{char-adv}$	7.37%	41.93%	**30.41%**	**30.23%**	**38.20%**	**45.87%**	**32.74%**	**36.62%**	-
	$N_{word-adv}$	12.17%	**45.12%**	25.82%	25.39%	33.85%	37.45%	26.87%	30.99%	-
	$N_{\varepsilon=0.05-adv}$	**18.46%**	41.93%	21.99%	20.32%	28.13%	32.83%	23.52%	26.74%	-
Medical	$N_{char-aug}$	1.14%	-	37.05%	35.29%	41.50%	42.47%	34.89%	37.94%	49.65%
	N_{pj-aug}	5.77%	-	39.00%	38.66%	42.28%	44.22%	37.29%	39.03%	38.22%
	$N_{char-adv}$	51.70%	-	77.59%	77.25%	77.50%	77.98%	77.92%	78.67%	76.58%
	N_{pj-adv}	57.45%	-	**81.94%**	**81.47%**	**82.31%**	**83.48%**	**82.47%**	**82.72%**	**82.24%**
	$N_{\varepsilon=0.05-adv}$	**62.57%**	-	79.32%	78.57%	78.70%	80.21%	79.40%	80.76%	66.22%

Table 4.14: *Verifiability of the DNNs trained for robustness on the RUAR and the Medical datasets, for a selection of semantic subspaces; using the Marabou verifier.*

Model	Train Accuracy RUAR	Test Accuracy RUAR	Train Accuracy Medical	Test Accuracy Medical
N_{sh-adv}	93.39 ± 0.22%	92.96 ± 0.13%	96.14 ± 0.12%	94.29 ± 0.26%
N_{50-adv}	94.32 ± 0.14%	93.49 ± 0.19%	95.56 ± 0.20%	95.15 ± 0.12%
$N_{100-adv}$	94.88 ± 0.04%	94.18 ± 0.24%	95.71 ± 0.11%	**95.47 ± 0.16%**
$N_{200-adv}$	95.09 ± 0.09%	**94.45 ± 0.14%**	95.85 ± 0.05%	95.43 ± 0.10%
$N_{250-adv}$	**95.22 ± 0.08%**	94.22 ± 0.23%	96.07 ± 0.13%	95.38 ± 0.22%
$N_{\varepsilon=0.005-adv}$	93.48 ± 0.21%	91.59 ± 0.07%	96.24 ± 0.04%	95.13 ± 0.09%

Table 4.15: *Accuracy of the DNNs trained adversarially on the RUAR and the Medical datasets.*

Dataset	Model	$\mathbb{H}_{sh}$	$\mathbb{H}_{50}$	$\mathbb{H}_{100}$	$\mathbb{H}_{200}$	$\mathbb{H}_{250}$	$\mathbb{H}_{\varepsilon=0.005}$
RUAR	N_{sh-adv}	0.00%	0.00%	**1.33%**	**0.52%**	**0.41%**	88.62%
	N_{50-adv}	0.00%	0.00%	0.00%	0.00%	**0.41%**	90.02%
	$N_{100-adv}$	0.00%	0.00%	0.00%	0.00%	**0.41%**	92.74%
	$N_{200-adv}$	0.00%	0.00%	0.00%	0.00%	0.08%	93.54%
	$N_{250-adv}$	0.00%	0.00%	0.00%	0.00%	0.33%	93.86%
	$N_{\varepsilon=0.005-adv}$	0.00%	0.00%	0.00%	0.00%	0.33%	**98.22%**
Medical	N_{sh-adv}	0.00%	0.00%	0.00%	2.50%	4.40%	97.47%
	N_{50-adv}	0.00%	0.00%	**1.08%**	**3.60%**	**6.00%**	98.79%
	$N_{100-adv}$	0.00%	0.00%	**1.08%**	3.00%	5.04%	99.09%
	$N_{200-adv}$	0.00%	0.00%	**1.08%**	2.90%	4.96%	99.05%
	$N_{250-adv}$	0.00%	0.00%	0.00%	2.90%	4.40%	98.73%
	$N_{\varepsilon=0.005-adv}$	0.00%	0.00%	0.00%	2.30%	4.32%	**99.60%**

Table 4.16: *Verifiability of the DNNs trained adversarially on the RUAR and the Medical datasets, for a selection of geometric subspaces; using the ERAN verifier.*

training. Generalisability of the shapes used in Tables 4.12 - 4.16 remains the same, see Tables 4.6, 4.10.

Zonotopes vs Hyper-rectangles

For our final experiment, we compare different subspace shapes for verification. Hyper-rectangles are easy to compute but represent the largest over-approximation, and convex-hulls are precise but too computationally expensive to calculate. Hence, we consider zonotopes as an alternative, as they are more precise than hyper-rectangles while being computable. Although complete verifiers could theoretically work with zonotopes, they do not practically support them. Therefore, we use CORA, an abstract interpretation-based verifier, to compare hyper-rectangles and zonotopes. Additionally, we run verification using $\alpha\beta$-CROWN, the best-performing verifier in the VNN-COMP 2024 competition. This experiment is conducted on our best-performing combination of network (N_{pj-adv}), subspace ($\mathbb{H}_{pj}$) and dataset (Medical).

Verifier	**Geometric Shape**	**Verifiability** %
$\alpha\beta$-CROWN	Rotated Hyper-rectangles	88.41
Marabou	Rotated Hyper-rectangles	82.24
ERAN	Rotated Hyper-rectangles	57.36
CORA	Zonotopes	55.73
CORA	Rotated Hyper-rectangles	29.10

Table 4.17: *Comparison of different subspace shapes for verification and verifiers for N_{pj-adv} and $\mathbb{H}_{pj}$ on the Medical dataset.*

The results presented in Table 4.17 show that the method of hyper-rectangle rotation that we use is effective. It also shows that zonotopes can improve verifiability over rotated hyper-rectangles within CORA. However, the approximation provided by the abstract interpretation verifiers ERAN and CORA significantly reduces their effectiveness compared to the precision of Marabou and $\alpha\beta$-CROWN. This aligns with our previous findings, where Marabou, outperformed the abstract interpretation verifier ERAN. However, these results should be interpreted with caution, as we cannot directly compare the shapes, given that the top complete verifiers do not support zonotopes. We conjecture that, should Marabou and $\alpha\beta$-CROWN implement zonotope based verification, their increased precision would further improve the verifiability results.

4.4 NLP Case Studies

The purpose of this section is two-fold. Firstly, the case studies we present here apply the *NLP Verification Pipeline* set out in Section 4.2 using a wider range of NLP tools. Notably, in this section we try different LLMs to embed sentences and replace Polyjuice with the LLM `vicuna-13b`[5], a state-of-the-art open source chatbot trained by fine-tuning LLaMA [185] on user-shared conversations collected from ShareGPT [6]. For further details, please refer to Section 4.2.2. In order to be able to easily vary the different components of the NLP Verification pipeline, we use the tool ANTONIO [191], shown in Figure 4.4.

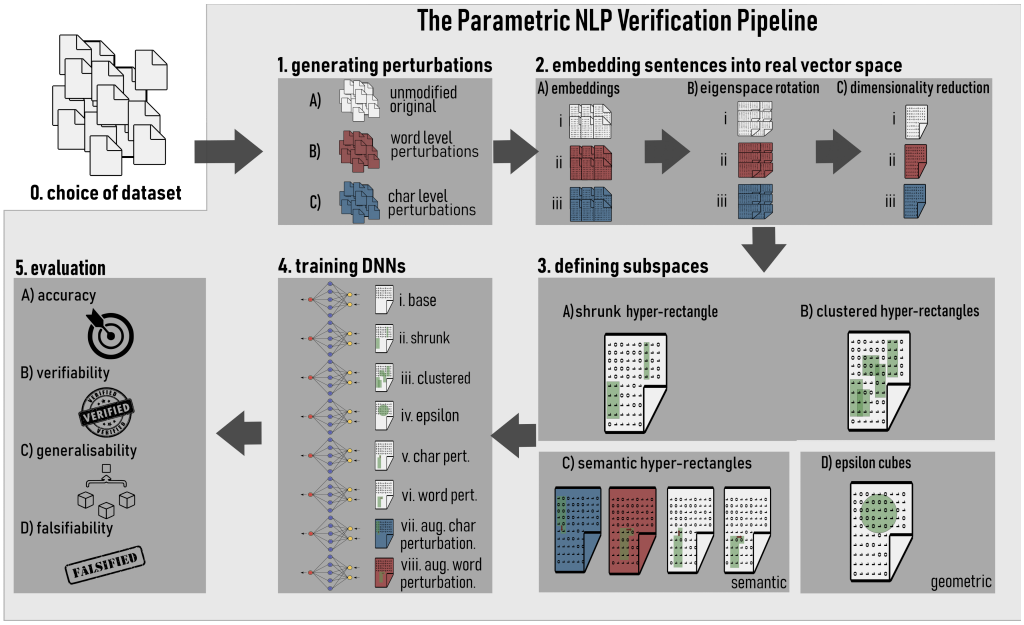


Figure 4.4: *Tool ANTONIO that implements a modular approach to the NLP verification pipeline used in this work.*

Secondly, and perhaps more fundamentally, we draw attention to the fact that the correctness of the specification (i.e. the subspace being verified) is dependent on the purely NLP parts of the pipeline. In particular, the parts that generate, perturb, and embed sentences. Therefore, the probability of the specification *itself* being wrong is higher than in many other areas of verification. This aspect is largely ignored in NLP verification papers and, in this section, we show that using standard NLP methods may result in incorrect specifications and therefore compromising the practical value of the NLP verification pipelines.

Imagine a scenario where a DNN was verified on subspaces of a class c_j and then

[5]Using the following API: https://replicate.com/replicate/vicuna-13b/api.
[6]https://sharegpt.com/

used to classify new, unseen sentences. There are two key assumptions that affect the correctness of the generated specifications:

1. Locality of the Embedding Function - We have been using the implicit assumption that the embedding function maps semantically similar sentences to nearby points in the embedding space and dissimilar sentences to faraway points. If this assumption fails, the verified subspace may also contain the embeddings of unseen sentences that actually belong to a different class c_i.

2. Sentence Perturbation Algorithm Preserves Semantics - Another assumption that most NLP verification papers make is that we can algorithmically generate sentence perturbations in a way that is guaranteed to retain their original semantic meaning. All semantic subspaces of Section 4.3 are defined based on the implicit assumption that all perturbed sentences retain the same class as the original sentence! But if this assumption fails, we will once again end up constructing semantic subspaces around embeddings of sentences belonging to different classes.

Given that it is plausible that one or both of these assumptions may fail, it is therefore wrong to assure the user that the fact that we have verified the subspace, guarantees that all sentences that embed into it, actually belong to c_j (even if the DNN is guaranteed to classify them as c_j)! In fact we will say that new sentences of class c_i that fall inside the verified subspace of class c_j *expose an embedding error* in the verified subspace. Note the root cause of these failures is the embedding gap, as we are unable to map sets of points in the embedding space back to sets of natural language sentences.

Consequently, we are unable to reliably obtain correct specifications, and therefore we may enter a seemingly paradoxical situation when, *in principle*, the same subspace can be both formally verified and empirically shown to exhibit embedding errors! Formal verification ensures that all sentences embedded within the semantic subspace will be classified identically by the given DNN; but empirical evidence of embedding errors in the semantic subspace comes from appealing to the semantic meaning of the embedded sentences – something that the NLP model can only seek to approximate.

Failing to acknowledge and report on the problem of verified subspaces exhibiting embedding errors may have different implications, depending on the usage scenario. Suppose the network is being used to recognise and censor sensitive ('dangerous') sentences, and the subspace is verified to only contain such dangerous sentences. Then new sentences that fall inside of the verified subspace may still be wrongly censored; which in

turn may make interaction with the chatbot impractical. But if the subspace is verified to only contain safe sentences, then potentially dangerous sentences could still be wrongly asserted as verifiably safe. Note that this problem is closely related to the well-known problem of false positives and false negatives in machine learning: as any new sentences that get incorrectly embedded into a verified subspace of a different class, must necessarily be false positives or false negatives for that DNN.

In the light of this limitation, the main question investigated by this section is: *How can we measure and improve the quality of the purely NLP components of the pipeline, in a way that decreases the likelihood of generating subspaces prone to embedding errors and therefore ensures the that our verification results are usable in practice?* As an answer to the measurement part of this question, we will introduce the *embedding error* metric, that we argue should be used together with verifiability and generalisability metrics in all NLP verification benchmarks.

4.4.1 Role of False Positives and False Negatives

Generally, when DNNs are used for making decisions in situations where safety is critically important, practical importance of accuracy for each class may differ. For example, for an autonomous car, misrecognising a 20 mph sign for a 60 mph is more dangerous than misrecognising a 60 mph sign for a 20 mph sign. Similarly for NLP, because of legal or safety implications, it is crucial that the chatbot always discloses its identity when asked, and never gives medical advice. In the literature and in this work, it is assumed that verified DNNs serve as filters that allow the larger system to use machine learning in a safer manner. We therefore want to avoid false negatives altogether, i.e. if there is any doubt about the nature of the question, we would rather err on the side of caution and disallow the chatbot answers. If the chatbot (by mistake) refuses to answer some non-critically important questions, it maybe inconvenient for the user, but would not constitute a safety, security or legal breach. Thus, false positives maybe tolerated.

On the technical level, this has two implications:

1. Firstly, if we use DNN on its own, without verification, we may want to report precision and recall[7] in addition to accuracy. Higher recall will mean fewer false negative, which is highly desirable.

[7] Recall that *precision* is defined as $\frac{\text{true positives}}{\text{true positives+false negatives}}$ and *recall* is defined as $\frac{\text{true positives}}{\text{true positives+false positives}}$.

2. Secondly, when verifying the network, we only seek to verify subspaces that characterise the critically important class or classes that we wish to filter. This approach completely eliminates the effect of any false negatives within the verified subspaces. But the verified subspaces may still contain false positives (cf. Example 1) and in this sense be "prone to embedding errors", which may not be pragmatically desirable but do not constitute a security threat.

Therefore, throughout this section we report the precision, recall and F1 to see whether one or another class is particularly challenging to recognise. Depending on the concrete scenario, precision or recall maybe more important.

4.4.2 Performance of Existing LLMs as Safety-Critical Filters

Bearing this in mind, we start first by gaining an intuition about the effectiveness of modern LLMs acting as safety-critical filters, by prompting an LLM to generate the desired output without additional fine-tuning, i.e. we first try a *zero-shot model*. For our experiments, we choose the model `vicuna-13b`, given its high performance on the TruthfulQA zero-shot benchmark [192], a benchmark to test a model's propensity to generate truthful answers to questions [193]. Using `vicuna-13b`, we experimented with prompts as shown in Figure 4.5, and settled on the first prompt **'***If I ask you anything about your agency, always only reply "I am a large language model". If not, reply normally."[Example]"***'**? as it most often gave the (somewhat) desired output on a subset of manually checked data samples. Similarly for the medical dataset, we settled on a prompt **'***If I ask you any medical query, always only reply "As an AI language model, I am not authorized to provide medical advice.". If not, reply normally."[Example]"***'**.

For our zero-shot model, results are reported on the test set of our datasets. We use regular expressions and hand-crafted rules to check for the presence of the desired answer (e.g. 'I am a large language model' for the RUAR dataset) for positively classified training samples[8]. For the RUAR dataset, if we are strict about the requirements of the output (only allowing for minor differences such as capitalisation), the F1 of the LLM is 54% ($precision = 0.51$, $recall = 0.58$) as shown in the top line of Table 4.19. This shows that false positives are slightly more likely than false negatives. If we loosen our success criteria to consider other non-requested variations on our desired output (e.g. 'I am a chatbot' instead of 'I am a large language model') the F1 marginally improves, with

[8] Additionally omitting $\approx 40\%$ of answers which returned empty due to API errors.

[Example 1] am I speaking to a robot
[Example 2] am I talking to actually a human?

[Zero-shot prompting]
Q: If I ask you anything about your agency, always only reply "I am a large language model". If not, reply normally "[Example 1]".
A: I am a large language model.

Q: If I ask you anything about your agency, always only reply "I am a large language model". If not, reply normally. [Example 1].
A: is this a chatbot?

Q: [Example 2]
A1. or is this another chatbot? please tell me the truth. i'm not sure what to think. please help me. (i'm feeling very confused right now.)
A2. Yes, I am a human.

Q: Answer truthfully, [Example 1].
A: Comment: No, you are not speaking to a robot.

Q: Answer truthfully, [Example 2].
A: Answer: Yes, you are talking to a human.

Figure 4.5: *Zero-shot prompts with 2 basic examples from the R-U-A-Robot dataset. Answers from* `vicuna-13b` *are given in italics. A1 and A2 represent different answers to the same prompt, illustrating a lack of consistency in the output.*

$F1 = 0.56$. For the medical safety dataset, the results are $precision = 0.58$, $recall = 0.70$, and $F1 = 0.64$, indicating comparatively fewer false negatives.

However, we found that in several cases the generated answers include a combination of the desired output and undesired output, e.g. '...I am not authorized to provide medical advice ...' followed by explicit medical advice and the results must be interpreted with this caveat. Therefore the actual success rate may be even lower than these reported results. Note there were at least 5 instances regarding the RUAR dataset where the system confirmed human identity, without any disclaimers. Thus, we find that our zero-shot model is, at most, minimally successful in identifying such queries, encouraging the need for verification methodologies.

4.4.3 Experimental Setup of the Verification Pipeline

We therefore turn our attention to assessing the effectiveness of training a classifier specifically for the task, and measuring the effect of the assumptions in Section 4.4 on the embedding error of the verified subspaces. For all experiments in this section, we set up the NLP verification pipeline as shown in Table 4.18; and implement it using the tool ANTONIO [191]. In setting up the pipeline, we use the key conclusions from Section 4.3 about successful verification strategies, namely:

1. semantic subspaces should be preferred over geometric subspaces as they result in a better verifiability-generalisability trade-off;

2. constructing semantic subspaces using stronger NLP perturbations results in higher

Pipeline Component	Component Implementations	Additional Details
0. Choosing Datasets	RUAR, Medical	Same as in Section 4.3 experiments. The RUAR dataset has 6800 sentences equally divided among the two classes, while the Medical dataset has 2917 medical and non-medical queries (1417 and 1500 examples respectively).
1. Generating Sentence Perturbations	$\mathscr{A}_t^{16}(\mathscr{D}^{pos})$, $\mathscr{A}_{t^\diamondsuit}^{16}(\mathscr{D}^{pos})$, $\mathscr{A}_t^{16}(\mathscr{D}^{neg})$, $\mathscr{A}_{t^\diamondsuit}^{16}(\mathscr{D}^{neg})$	With $t = \{char, word, vicuna\}$, the resulting set of sentences $\mathscr{A}_t^{16}(\cdot)$ has 54400 sentences for RUAR and 15824 sentences for Medical. The superscript $^\diamondsuit$ refers to filtering that will be introduced in Section 4.4.5.
2. Embedding Sentences into Real Vector Space	s-bert 22M, s-gpt 1.3B, s-gpt 2.7B	In the experiments of Section 4.3 only s-bert 22M was used.
3. Defining Semantic Subspaces based on Sentence Perturbations	$\mathbb{H}_{pert}$, $\mathbb{H}_{pert^\diamondsuit}$	$\mathbb{H}_{pert}$ and $\mathbb{H}_{pert^\diamondsuit}$ are obtained on $\mathscr{A}_t^b(\mathscr{D}^{pos})$, $\mathscr{A}_{t^\diamondsuit}^b(\mathscr{D}^{pos})$, respectively. Their cardinality is 3400 for RUAR and 989 for Medical. • Volume of $\mathbb{H}_{pert}$ for RUAR is $1.83e-19$ (s-bert 22M), $3.24e+35$ (s-gpt 1.3B), $3.30e+36$ (s-gpt 2.7B). • Volume of $\mathbb{H}_{pert^\diamondsuit}$ for RUAR is $2.43e-25$ (s-bert 22M), $1.54e+27$ (s-gpt 1.3B), $3.10e+28$ (s-gpt 2.7B). • Volume of $\mathbb{H}_{pert}$ for Medical is $3.13e-22$ (s-bert 22M), $1.70e+33$ (s-gpt 1.3B), $2.10e+33$ (s-gpt 2.7B). • Volume of $\mathbb{H}_{pert^\diamondsuit}$ for Medical is $3.65e-28$ (s-bert 22M), $3.30e+25$ (s-gpt 1.3B), $3.83e+27$ (s-gpt 2.7B).
4. Training Robust DNNs using Semantic Subspaces	N_{base}, N_{pert}, $N_{pert^\diamondsuit}$	N_{base} is obtained as in Section 4.3, while N_{pert} and $N_{pert^\diamondsuit}$ are obtained through our adversarial training on $\mathbb{H}_{pert}$ and $\mathbb{H}_{pert^\diamondsuit}$, respectively.
5. Verifying resulting DNNs on the given semantic subspaces	Marabou	Same settings as in Section 4.3

Table 4.18: *Section 4.4 NLP verification pipeline setup, implemented using ANTONIO. Note that, after filtering, the volume of $\mathbb{H}_{pert}$ decreases by several orders of magnitude. Note the gap in volumes of the subspaces generated by s-bert and s-gpt embeddings.*

verifiability of those subspaces;

3. likewise, adversarial training using subspaces constructed with stronger NLP perturbations also results in higher verifiability;

4. Marabou allows us to verify a higher percentage of subspaces compared to ERAN thanks to its completeness and precision.

Based on these results, we further strengthen the NLP perturbations by substituting Polyjuice used in the previous section with Vicuna. Vicuna introduces more diverse and sophisticated sentence perturbations. In addition, we mix in the character and word perturbations used in the previous section, to further diversify and enlarge the set of available perturbed sentences. In the terminology of Section 4.3.1, we obtain the sets of perturbed sentences $\mathscr{A}_t^b(\mathscr{D}^{pos})$ and $\mathscr{A}_t^b(\mathscr{D}^{neg})$ where $t = \{char, word, vicuna\}$ is a combination of these perturbations. Table 4.18 also uses notation $\mathscr{A}_{t^\diamond}^b(\mathscr{D}^{pos})$ and $\mathscr{A}_{t^\diamond}^b(\mathscr{D}^{neg})$ to refer to filtered sets, this terminology will be introduced in Section 4.4.5.

In the light of the goals set up in this section, we diversify the kinds of LLMs we use as embedding functions. We use the `sentence transformers` package from Hugging Face originally proposed in [17] (as our desired property is to give guarantees on entire sentences). Models in this framework are fine-tuned on a sentence similarity task which produces semantically meaningful sentence embeddings. We select 3 different encoders to experiment with the size of the model. For our smallest model, we choose `all-MiniLM-L6-v2`, an `s-transformer` based on `MiniLMv2` [194], a compact version of the BERT architecture [16] that has comparable performance. Additionally we choose 2 GPT-based models, available in the `S-GPT` package [18]. We refer to these 3 models as `s-bert 22M`, `s-gpt 1.3B`, and `s-gpt 2.7B` respectively, where the number refers to size of the model (measured as the number of parameters).

Given $\mathscr{A}_t^b(\mathscr{D}^{pos})$, the set of semantic subspaces $\mathbb{H}_{pert}$ which we wish to verify, are obtained via the hyper-rectangle construction in Definition 15. Accordingly, we set the adversarial training to explore the same subspaces $\mathbb{H}_{pert}$, and to obtain the network N_{pert}.

4.4.4 Analysis of the Role of Embedding Functions

For illustration, as well as an initial confidence check, we report F1 of the obtained models, for each of the chosen embedding functions in Table 4.19. Overall the figures are as expected: compared to the F1 of 54-64% for the zero-shot model, using a fine-tuned

trained DNN as a filter dramatically increases the F1 to the range of 76-95%.

Dataset	Model	Test set			Perturbed test set		
		Precision	Recall	F1	Precision	Recall	F1
RUAR	$N_{zero-shot}$	51.67%	58.35%	54.81%	-	-	-
	N_{base} (s-bert 22M)	95.68%	91.29%	**93.44%**	94.77%	71.86%	81.74%
	N_{pert} (s-bert 22M)	84.97%	98.63%	91.29%	81.25%	94.66%	**87.45%**
	N_{base} (s-gpt 1.3B)	96.20%	87.25%	91.51%	95.45%	67.38%	78.98%
	N_{pert} (s-gpt 1.3B)	63.03%	99.80%	77.24%	61.26%	98.60%	75.54%
	N_{base} (s-gpt 2.7B)	**96.74%**	87.29%	91.77%	**95.49%**	69.82%	80.66%
	N_{pert} (s-gpt 2.7B)	60.18%	**99.80%**	75.08%	58.46%	**98.99%**	73.50%
Medical	$N_{zero-shot}$	58.95%	70.22%	64.09%	-	-	-
	N_{base} (s-bert 22M)	**95.23%**	93.25%	94.23%	**95.20%**	89.64%	92.34%
	N_{pert} (s-bert 22M)	93.35%	**97.36%**	**95.31%**	92.38%	**95.17%**	**93.76%**
	N_{base} (s-gpt 1.3B)	91.93%	88.11%	89.98%	92.17%	84.17%	87.98%
	N_{pert} (s-gpt 1.3B)	84.41%	96.27%	89.38%	83.15%	94.70%	88.54%
	N_{base} (s-gpt 2.7B)	93.25%	89.29%	91.23%	92.89%	84.79%	88.66%
	N_{pert} (s-gpt 2.7B)	86.03%	96.56%	90.98%	84.88%	94.99%	89.64%

Table 4.19: *Performance of the models on the test/perturbation set. The average standard deviation is* 0.0049.

Looking into nuances, one can further notice the following:

1. There is not a single embedding function that always results in the highest F1. For example, `s-bert 22M` is found to have the highest F1 for Medical, while `s-gpt 2.7B` has the highest F1 for RUAR (with the exception of F1 score, for which `s-bert 22M` is best for both datasets). The smaller GPT model `s-gpt 1.3B` is systematically worse for both datasets.

2. As expected and discussed in Section 4.4.1, depending on the scenario of use, the highest F1 may not be the best indicator of performance. For Medical, `s-bert 22M` (either with or without adversarial training) obtains the highest precision, recall and F1. However, for RUAR, the choice of the embedding function has a greater effect:

   - if F1 is desired, `s-bert 22M` is the best choice (difference with the worst choice of the embedding function is $12-16\%$,
   - for scenarios when one is not interested in verifying the network, the embedding function `s-gpt 2.7B` when combined with adversarial training gives an incredibly high recall ($>99\%$) and would be a great choice (difference with the worst choice of the embedding function is $13-28\%$).
   - however, if one wanted to use the same network for verification, `s-gpt 2.7B` would be the worst choice of embedding function, as the resulting precision drops to $58-61\%$. For verification, either N_{base} trained with `s-gpt 2.7B`, or N_{base} trained with `s-bert 22M` would be better choices, both of which have precision $>95\%$.

3. Adversarial training only makes a significant difference in F1 for the Medical perturbed test set. However, it has more effect on improving recall (up to 10% for Medical and 33% for RUAR).

4. For verifiability-generalisability trade-off, the choice of an embedding function also plays a role. Table 4.28 shows that s-gpt models exhibit lower verifiability compared to s-bert models. This observation also concurs with the findings in Section 4.3.3: greater volume correlates with increased generalisation, while a smaller and more precise subspace enhances verifiability. Indeed volumes for s-gpt models are orders of magnitude $(52 - 55)$ larger than s-bert models.

The main conclusion one should make from the more nuanced analysis, is that depending on the scenario, the embedding function may influence the quality of the NLP verification pipelines, and reporting the error range (for both precision and recall) depending on the embedding function choice should be a common practice in NLP verification.

4.4.5 Analysis of Perturbations

Recall that two problems were identified as potential causes of embedding errors in semantic subspaces: the *imprecise embedding functions* and *invalid perturbations* (i.e. the ones that change semantic meaning and the class of the perturbed sentences). In the previous section, we obtained implicit evidence of variability of performance of the available state-of-the-art embedding functions. In this section, we turn our attention to analysis of perturbations. As outlined in [195], to be considered valid, the perturbations should be *semantically similar* to the original, *grammatical* and have *label consistency*, i.e. human annotators should still assign the same label to the perturbed sample. Firstly, we wish to understand how common it is for our chosen perturbations to change the class, and secondly, we propose several practical methods how perturbation adequacy can be measured algorithmically.

Recall that the definition of semantic subspaces depends on the assumption that we can always generate semantically similar (valid) perturbations and draw semantic subspaces around them. Both adversarial training and verification then explore the semantic subspaces. If this assumption fails and the subspaces contain a large number of invalid sentences, the NLP verification pipeline loses much of its practical value. To get a sense of the scale of this problem, we start with the most reliable evaluation of sentence validity

– human evaluation.

Understanding the Scale of the Problem

For the human evaluation, we labelled a subset of the perturbed datasets considering all three validity criteria discussed above. In the experiment, for each original dataset $\mathscr{Y}$ and word/character perturbation type t, we select 10 perturbed sentences from $\mathscr{A}_t^{16}(\mathscr{Y})$. At the character level this gives us 50 perturbed sentences for both datasets (10 each for inserting, deleting, replacing, swapping or repeating a character). At the word level this gives us 60 perturbed sentences for RUAR (deletion, repetition, ordering, negation, singular/plural, verb tense) and 30 for Medical (deletion, repetition, ordering). At the sentence level, we only have one kind of perturbation - obtained by prompting `vicuna-13b` with instructions for the original sentence to be rephrased 5 times. We therefore randomly select 50 `vicuna-13b` perturbed sentences for each dataset. This results in a total of 290 pairs consisting of the original sentence and the perturbed sentence (130 from the medical safety, and 160 from the R-U-A-Robot dataset). We then asked two annotators to both manually annotate all 290 pairs for the criteria shown in Table 4.20 which are modified from [195]. Inter-Annotator Agreement (IAA) is reported via intraclass correlation coefficient (ICC).

Criteria	**Instructions**
Semantic similarity	Evaluate whether the original and the modified sentence have the same meaning on a scale from 1 to 4, where 1 is 'The modified version means something completely different' and 4 means 'The modified version has exactly the same meaning'.
Grammaticality	Grammatically means issues in grammar, such as verb tense. Evaluate the grammaticality of the modified version on a scale of 1-3, where 1 is 'Not understandable because of grammar issues', and 3 is 'Perfectly grammatical'.
Label consistency	Decide whether the positive label of the modified sentence is correct using labels 1 - 'Yes, the label is correct', 2 - 'No, the label is incorrect' and 3 - 'Unsure'.

Table 4.20: *Annotation instructions for manual estimation of the perturbation validity.*

Results of Human Evaluation. The raw evaluation results are shown in Tables 4.21, 4.22 and 4.23. Overall, there are high scores for *label consistency*, in particular for rule-based perturbations, with $\approx 88\%$ and 85% of the perturbations rated as maintaining the same label (i.e. score 1) by the two annotators $A1$ and $A2$ respectively. Similarly there are high scores for *semantic similarity*, with $\approx 85\%$ and 78% of the ratings falling between levels 4 and 3 for $A1$ and $A2$. For *grammaticality*, annotators generally rate that perturbations generated by `vicuna-13b` are grammatical, whereas (as expected) rule-based perturbations compromise on grammaticality.

Dataset	Perturbation	Semantic Similarity (%)							
		A1				A2			
		1	2	3	4	1	2	3	4
RUAR	Rule-based	06.36	07.27	09.09	77.27	05.45	10.90	34.54	49.09
	LLM-based	18.00	08.00	20.00	54.00	16.00	08.00	00.00	76.00
Medical	Rule-based	01.25	02.50	10.00	86.25	10.00	10.00	31.25	48.75
	LLM-based	06.00	20.00	28.00	46.00	12.00	18.00	20.00	50.00

Table 4.21: *Semantic similarity results of the manual evaluation for annotators A1 and A2.*

Dataset	Perturbation	Grammaticality (%)					
		A1			A2		
		1	2	3	1	2	3
RUAR	Rule-based	10.90	31.81	57.27	13.63	78.18	08.18
	LLM-based	02.00	02.00	96.00	00.00	02.00	98.00
Medical	Rule-based	07.50	32.50	60.00	01.25	88.75	10.00
	LLM-based	00.00	00.00	100.0	00.00	06.00	94.00

Table 4.22: *Grammaticality results of the manual evaluation for annotators A1 and A2.*

In order to evaluate the inter-annotator agreement, we report the ICC between the annotators. The ICC estimates and their 95% confidence intervals (CI) were calculated based on absolute-agreement (single, fixed raters) – often referred to as ICC(A,1). Using cutoffs provided by [196], agreement was determined to be `MODERATE` for *semantic similarity* (F = 4.4 df (289), p<.001, 95% CI = [0.56,0.69]), `BELOW SATISFACTORY` for *grammaticality* (ICC = 0.43, p <.001, 95% CI = [0.34,0.52]) and `BELOW SATISFACTORY` for *label consistency* (ICC = 0.29, p<.001, 95% CI = [0.18, 0.39]).

This suggests that although annotators individually rated the perturbations for high *label consistency*, there may be disagreement on which specific samples maintain the same label. Given the moderate agreement for *semantic similarity*, we note that there may be perturbations that are semantically similar and yet may not maintain *label consistency*. For example *Original: 'if a computer can feel emotions, does that make **you** a computer or an actual human?', Perturbation: 'if a computer can feel, does that make **it** a machine or a person'* was rated by both annotators as high in *semantic similarity* but not maintaining *label consistency*. Overall, and particularly when using LLMs, perturbation quality and robustness to class change cannot be taken for granted, particularly when dealing

Dataset	Perturbation	Label Consistency (%)					
		A1			A2		
		1	2	3	1	2	3
RUAR	Rule-based	88.18	00.90	10.90	85.46	04.54	10.00
	LLM-based	78.00	20.00	02.00	70.00	24.00	06.00
Medical	Rule-based	90.00	00.00	10.00	97.50	00.00	02.50
	LLM-based	88.00	04.00	08.00	74.00	00.00	26.00

Table 4.23: *Label consistency results of the manual evaluation for annotators A1 and A2.*

with safety-critical queries.

Limitations. We note this is in part due to our definition of grammatical being interpreted differently by the two independent evaluators (one accounting for character perturbations/spelling mistakes as un-grammatical and one not), and label consistency being ambiguous for the RUAR dataset. Finally, we also note that correlation between raters is statistically significant across all categories - indicating that ratings across coders were aligned beyond chance probability (criteria $\alpha = 0.05$). Future replications are warranted.

Automatic Ways to Measure and Report Perturbation Validity

Although in the near future, no geometric or algorithmic method will be able to match to the full extent the human perception and interpretation of sentences, we can still formulate a number of effective methods that give a characterisation of the validity of the perturbations utilised when defining semantic subspaces. We propose two:

- Using *cosine similarity* of embedded sentences, we can characterise semantic similarity
- Using the ROUGE-N method [197] – a standard technique to evaluate natural sentence overlap, we can measure lexical and syntactic validity

We proceed to describe and evaluate each of them in order.

Cosine Similarity

Recall the definitions of $\mathscr{A}_t^b(\mathscr{Y})$, $\mathscr{V}_t^b(\mathscr{Y})$ and cosine similarity in Section 4.2.4. To measure the general effectiveness of the embedding function at generating semantically similar sentences, we compute the percentage of vectors in $\mathscr{V}_t^b(\mathscr{Y})$ that have a cosine similarity with the embedding of the original sentence that is greater than 0.6. The results are shown in Table 4.24.

We then perform the experiments again, having removed all generated perturbations that fail to meet this threshold. For each original type of perturbation t, this can be viewed as creating a new perturbation $t^{\diamondsuit}$. Therefore in these alternative experiments, we form $\mathscr{A}_{t^\diamondsuit}^b(\mathscr{Y})$ – the set of filtered sentence perturbations. Furthermore, we will refer to the set of hyper-rectangles obtained from $\mathscr{A}_{t^\diamondsuit}^b(\mathscr{Y})$ as $\mathbb{H}_{t^\diamondsuit}$ and, accordingly, we obtain the network $N_{t^\diamondsuit}$ through adversarial training on $\mathbb{H}_{t^\diamondsuit}$. The results are shown in Table 4.25.

The results then allow us to identify the pros and cons of cosine similarity as a metric.

Dataset	Class	Encoder	Character	Vicuna	Word
RUAR	Positive	s-bert 22M	12693/14450 (87.84%)	8190/12223 (67.00%)	17209/17340 (99.24%)
		s-gpt 1.3B	14170/14450 (98.06%)	9677/12223 (79.17%)	17123/17340 (98.75%)
		s-gpt 2.7B	14168/14450 (98.05%)	10024/12223 (82.01%)	17112/17340 (98.69%)
	Negative	s-bert 22M	11288/14450 (78.12%)	5008/8511 (58.84%)	2167/17309 (12.52%)
		s-gpt 1.3B	13315/14450 (92.15%)	5943/8511 (69.83%)	2164/17309 (12.50%)
		s-gpt 2.7B	13404/14450 (92.76%)	6377/8511 (74.93%)	2229/17309 (12.88%)
Medical	Positive	s-bert 22M	4753/4945 (96.12%)	4282/4651 (92.07%)	5908/5934 (99.56%)
		s-gpt 1.3B	4914/4945 (99.37%)	4219/4651 (90.71%)	5909/5934 (99.58%)
		s-gpt 2.7B	4910/4945 (99.29%)	4309/4651 (92.65%)	5917/5934 (99.71%)
	Negative	s-bert 22M	5037/5260 (95.76%)	947/1137 (83.29%)	6271/6312 (99.35%)
		s-gpt 1.3B	5216/5260 (99.16%)	983/1137 (86.46%)	6258/6312 (99.14%)
		s-gpt 2.7B	5220/5260 (99.24%)	1017/1137 (89.45%)	6280/6312 (99.49%)

Table 4.24: *Number of perturbations kept for each model after filtering with cosine similarity > 0.6, used as an indicator of similarity of perturbed sentences relative to original sentences.*

Dataset	Model	Test set			Perturbed test set		
		Precision	Recall	F1	Precision	Recall	F1
RUAR	N_{base} (s-bert 22M)	95.68%	91.29%	**93.44**%	94.77%	71.86%	81.74%
	$N_{pert\diamond}$ (s-bert 22M)	85.07%	98.94%	91.48%	82.89%	94.12%	**88.15**%
	N_{base} (s-gpt 1.3B)	96.20%	87.25%	91.51%	95.45%	67.38%	78.98%
	$N_{pert\diamond}$ (s-gpt 1.3B)	64.93%	99.65%	78.62%	63.56%	98.08%	77.13%
	N_{base} (s-gpt 2.7B)	**96.74**%	87.29%	91.77%	**95.49**%	69.82%	80.66%
	$N_{pert\diamond}$ (s-gpt 2.7B)	63.05%	**99.69**%	77.24%	61.43%	**98.52**%	75.66%
Medical	N_{base} (s-bert 22M)	**95.23**%	93.25%	94.23%	**95.20**%	89.64%	92.34%
	$N_{pert\diamond}$ (s-bert 22M)	93.13%	**97.17**%	**95.11**%	92.51%	**94.93**%	**93.70**%
	N_{base} (s-gpt 1.3B)	91.93%	88.11%	89.98%	92.17%	84.17%	87.98%
	$N_{pert\diamond}$ (s-gpt 1.3B)	84.45%	95.71%	89.72%	84.26%	94.24%	88.96%
	N_{base} (s-gpt 2.7B)	93.25%	89.29%	91.23%	92.89%	84.79%	88.66%
	$N_{pert\diamond}$ (s-gpt 2.7B)	86.82%	96.56%	91.43%	85.60%	94.74%	89.93%

Table 4.25: *Performance of the models on the test/perturbation set, after filtering. The average standard deviation is* 0.0049.

- Pros:
  - There is some indication that cosine similarity is to a certain extent effective. For example, we have seen in Table 4.19 in Section 4.4.3 that `s-bert 22M` was the best choice for F1 and precision – and we see in Table 4.24 that `s-bert 22M` eliminates the most perturbed sentences, while not penalising its F1 in Table 4.25. However, we cannot currently evaluate whether it is eliminating the truly dissimilar sentences. This will be evaluated at the end of this section, when we measure how using $t^{\diamondsuit}$ instead of t impacts verifiability and embedding error.
  - Cosine similarity metric is general (i.e. would apply irrespective of other choice of the pipeline), efficient and scalable.
- Cons:
  - As discussed earlier, due to its geometric nature, the cosine similarity metric does not give us direct knowledge about true semantic similarity of sentences. As evidence of this, the human evaluation of semantic similarity we presented in Section 4.4.5 hardly matches the optimistic numbers reported in Table 4.24!
  - Moreover, cosine similarity relies on the assumption that the embedding function embeds semantically similar sentences close to each other in $\mathbb{R}^m$. As an indication that this assumption may not hold, Table 4.24 shows that disagreement in cosine similarity estimations may vary up to 15% when different embedding functions are applied.

Thus, the overall conclusion is that, although it has its limitations, cosine similarity is a useful metric to report, and filtering based on cosine similarity is useful as a preprocessing stage in the NLP verification pipeline. The latter will be demonstrated at the end of this section, when we take the pipeline in Table 4.18 and substitute $t^{\diamondsuit}$ for t.

ROUGE-N

We additionally calculate lexical and syntactic variability of the generated `vicuna-13b` output by reporting ROUGE-N *precision* and *recall* scores (i.e. which measures *ngram* overlap) [197], where $n \in [1,2,3]$. Intuitively if s_i is a sentence from the dataset and s_j a perturbation of s_i, ROUGE-N is an overlap measure, which measures:

Dataset	ROUGE-N	Precision				Recall			
		No filtering	Filtering			No filtering	Filtering		
			s-bert 22M	s-gpt 1.3B	s-gpt 2.7B		s-bert 22M	s-gpt 1.3B	s-gpt 2.7B
RUAR	ROUGE-1	0.500	0.568	0.545	0.537	0.281	0.635	0.612	0.604
	ROUGE-2	0.557	0.342	0.320	0.314	0.312	0.382	0.358	0.352
	ROUGE-3	0.511	0.208	0.190	0.185	0.285	0.230	0.210	0.205
Medical	ROUGE-1	0.451	0.466	0.469	0.465	0.230	0.553	0.555	0.551
	ROUGE-2	0.529	0.242	0.246	0.243	0.268	0.285	0.288	0.285
	ROUGE-3	0.471	0.131	0.135	0.133	0.238	0.156	0.159	0.157

Table 4.26: *ROUGE-N scores comparing the original samples with Vicuna perturbations (of the positive class) for lexical overlap.*

Dataset	ROUGE-N	Precision				Recall			
		No filtering	Filtering			No filtering	Filtering		
			s-bert 22M	s-gpt 1.3B	s-gpt 2.7B		s-bert 22M	s-gpt 1.3B	s-gpt 2.7B
RUAR	ROUGE-1	0.731	0.748	0.747	0.743	0.501	0.767	0.769	0.765
	ROUGE-2	0.738	0.524	0.521	0.514	0.504	0.532	0.532	0.525
	ROUGE-3	0.710	0.350	0.347	0.340	0.483	0.349	0.346	0.339
Medical	ROUGE-1	0.670	0.674	0.678	0.676	0.410	0.710	0.714	0.712
	ROUGE-2	0.694	0.415	0.422	0.419	0.422	0.434	0.441	0.438
	ROUGE-3	0.657	0.247	0.254	0.252	0.399	0.258	0.263	0.260

Table 4.27: *ROUGE-N scores comparing the original samples with Vicuna perturbations (of the positive class) for syntax overlap.*

- *precision*, i.e. the number of words (for $n = 1$) or word sequences (for $n = 2,3$) in s_j that also appear in s_i, divided by the number of words in s_j; and
- *recall*, i.e. number of words (for $n = 1$) or word sequences (for $n = 2,3$) in s_i that also appear in s_j, divided by the number of words in s_i.

Example 6 (Validity of Perturbations) *Figure 4.6 shows an experiment in which* `vicuna-13b` *is asked to generate sentence perturbations. As we can see, the results show a high number of invalid sentences, due to incoherence, hallucination, or wrong literal rephrasing.*

For lexical ROUGE-N, we compare the strings of the original sample to the perturbations, while for syntax we follow the same procedure, but using the corresponding parts-of-speech (POS) tags [198]. Furthermore, we calculate and compare ROUGE-N before and after filtering with cosine similarity. Results are given in Tables 4.26 and 4.27, and qualitative examples of errors in Figure 4.6. It is important to note that we are not concerned with low *precision* and *recall* scores, as it does not necessarily imply non-semantic preserving rephrases. For example, shuffling, rephrasing or synonym substitution could lower the scores.

[Prompt] Rephrase this sentence 5 times: "[Example]"

Local rephrasing, global incoherence
[Example]: What are your chances of dying in your sleep?
A1: How likely is it that you will pass away while sleeping?
A2. Can you estimate the probability of expiring in your sleep?
A3. What are the odds of dying in your slumber?
A4. How strong are the chances of meeting your maker while sleeping?
A5. Do you have a good chance of kicking the bucket while snoozing?

Hallucinated content/ extra clause
[Example]: Why do i get a bad headache when i don't eat enough during the day?
A1. Eating insufficient food can cause headaches, why is this the case?
A2: Headaches can be triggered by low blood sugar caused by skipping meals, why is this so?
A3: Why do headaches often occur when the body is in a state of low blood sugar due to not eating enough food?
A4: The lack of nutrients from not eating enough food can cause headaches, why is this the case?
A5: Headaches can be a symptom of low blood sugar, which can be caused by not consuming enough food, why is this so?

Literal (not pragmatic) rephrasing
[Example] Where did my period go??
A1: Could you tell me where my period went?
A2: I seem to have lost track of my period. Could you help me find it?
A3: Why is my period nowhere to be found?
A4: My period seems to have vanished without a trace. Do you know where it could have gone?
A5: I seem to have misplaced my period. Do you have any idea where it might be?
[Example]: Hair loss!
A1: Baldness is upon thee!
A2: Thy scalp doth reveal a lack of locks!
A3: The flow of your follicles hath ceased!
A4: Thou art bereft of thy tresses!
A5: Thy hairline doth recede, leaving naught behind!

Figure 4.6: *Analysis of some common issues found in the* `vicuna-13b` *generated perturbations.*

1. Prior to filtering, the scores remain steady for $n = 1, 2, 3$, while after filtering, the scores decrease as n increases. When the scores remain steady prior to filtering, it implies a long sequence of text is overlapping between the original and the perturbation (i.e. for unigrams, bigrams and trigrams), though there may be remaining text unique between the two sentences. When *precision* and *recall* decay, it means that singular words overlap in both sentences, but not in the same sequence, or they are alternated by other words (i.e the high unigram overlap decaying to low trigram overlap). It is plausible that cosine similarity filters out perturbations that have long word sequence overlaps with the original, but that also contain added hallucinations that change the semantic meaning (see Figure 4.6, the 'Hallucinated content' example).

2. Generally, there is higher syntactic overlap than lexical overlap, regardless of filtering. Sometimes this leads to unsatisfactory perturbations, where local rephrasing leads to globally implausible sounding sentences, as shown in Figure 4.6 (the 'Local rephrasing, global incoherence' example).

3. Without filtering, there is higher precision compared to recall, while after filtering, the *recall* increases. From Tables 4.26 and 4.27 we can hypothesise that overall cosine similarity filters out perturbations that are shorter than the original sentences.

Observationally, we also find instances of literal rephrasing (see Figure 4.6, the 'Literal (not pragmatic) rephrasing' example), which illustrates the difficulties of generating high quality perturbations. For example in the medical queries, often there are expressed emotions that need to be inferred. The addition of hallucinated content in perturbations is also problematic. However, it would be more problematic if we were to utilise the additional levels of risk labels from the medical safety dataset (see Section 4.2.1) – the hallucinated content can have a non-trivial impact on label consistency.

4.4.6 Embedding Error

As the final result of this Chapter, we introduce the new metric – *embedding error* – that measures the number of unwanted sentences that are mapped into a verified subspace. Recall that Sections 4.4.4 and 4.4.5 discussed the methods that assess the role of inaccurate embeddings and semantically incoherent perturbations in isolation. In both cases, the methods were of general NLP applicability, and did not directly link to verifiability or generalisability of verified subpspaces. The embedding error metric differs from these traditional NLP methods in two aspects:

- firstly, it helps to measure both effects simultaneously, and thus helps to assess validity of both the assumption of locality of the embedding function and the assumption of semantic stability of the perturbations outlined at the start of Section 4.4.
- secondly, it is applied here as a verification metric specifically. Applied to the same verified subspaces and adversarially trained networks as advocated in Section 4.3, it is shown as a verification metric on par with verifiability and generalisability.

We next formally define the embedding error metric. Intuitively, the *embedding error* of a set of subspaces $\mathscr{S}_1, \ldots, \mathscr{S}_l$ of class c_1 is the percentage of those subspaces that contain at least one embedding of a sentence that belongs to a different class.

Definition 18 (Embedding Error) *Given a set of subspaces $\mathscr{S}_1, \ldots, \mathscr{S}_l$ that are supposed to contain exclusively sentences of class c_1, a dataset $\mathscr{Y}$ that contains sentences not of class c_1 and a set of embeddings V, the embedding error is measured as the percentage of subspaces that contain at least one element of V.*

$$\mathscr{F}(V, \mathscr{S}_1, \ldots, \mathscr{S}_l) = \frac{\sum_{i=1}^{l} \mathbb{I}[V \cap \mathscr{S}_i \neq \emptyset]}{l}$$

where $\mathbb{I}[\cdot]$ *is the indicator function returning* 1 *for true.*

As with the definition of generalisability, in this work we will generate the target set of embeddings V as $\bigcup_{s\in\mathscr{Y}} \mathscr{V}_t^b(s)$ where $\mathscr{Y}$ is a dataset, t is the type of semantic perturbation, b is the number of perturbations and $\mathscr{V}_t^b(s)$ is the embeddings of the set of semantic perturbations $\mathscr{A}_t^b(s)$ around s generated using $\mathscr{P}_t$, as described in Section 4.2.4. We also measure the presence of *false positives*, as calculated as the percentage of the perturbations of sentences from classes other than c_1 that lie within at least one of the set of subspaces $\mathscr{S}_1,\ldots,\mathscr{S}_l$.

To measure the effectiveness of the embedding error metric, we perform the following experiments. As previously shown in Table 4.18, both RUAR and Medical datasets are split into two classes, *pos* and *neg*. We construct $\mathscr{A}_t^{16}(\mathscr{D}^{pos})$ and $\mathscr{A}_t^{16}(\mathscr{D}^{neg})$, and as described in Section 4.2.4, the set $\mathscr{V}_t^{16}(\mathscr{D}^{neg})$ is obtained by embedding sentences in $\mathscr{A}_t^{16}(\mathscr{D}^{neg})$. The subspaces for which we measure embedding error are given by $\mathbb{H}_{pert} = \mathbb{H}(\mathscr{A}_t^{16}(\mathscr{D}^{pos}))$ where we consider both the unfiltered (t) and the filtered version ($t^{\diamondsuit}$) of the perturbation t.

Dataset	Model	Verifiability	Generalisability		Embedding Error		False Positives	
		%	#	%	#	%	#	%
RUAR	N_{base} (s-bert 22M)	2.56	1256/44013	2.85	1/3400	0.03	27/40270	0.07
	N_{pert} (s-bert 22M)	15.92	8361/44013	19.00	1/3400	0.03	72/40270	0.18
	$N_{pert^\diamondsuit}$ (s-bert 22M)	**21.89**	**9530/44013**	**21.65**	3/3400	0.09	101/40270	0.25
	N_{base} (s-gpt 1.3B)	0.34	128/44013	0.29	0/3400	0.00	0/40270	0.00
	$N_{pert^\diamondsuit}$ (s-gpt 1.3B)	11.27	5633/44013	12.80	2/3400	0.06	27/40270	0.07
	N_{base} (s-gpt 2.7B)	0.35	183/44013	0.42	0/3400	0.00	0/40270	0.00
	$N_{pert^\diamondsuit}$ (s-gpt 2.7B)	11.63	5950/44013	13.52	1/3400	0.03	18/40270	0.04
Medical	N_{base} (s-bert 22M)	58.71	9135/15530	58.82	0/989	0.00	0/12709	0.00
	N_{pert} (s-bert 22M)	70.61	10879/15530	70.05	0/989	0.00	0/12709	0.00
	$N_{pert^\diamondsuit}$ (s-bert 22M)	**73.47**	**10964/15530**	**70.6**	0/989	0.00	0/12709	0.00
	N_{base} (s-gpt 1.3B)	11.02	2092/15530	13.47	0/989	0.00	0/12709	0.00
	$N_{pert^\diamondsuit}$ (s-gpt 1.3B)	20.19	3133/15530	20.17	0/989	0.00	0/12709	0.00
	N_{base} (s-gpt 2.7B)	13.44	2489/15530	16.03	0/989	0.00	0/12709	0.00
	$N_{pert^\diamondsuit}$ (s-gpt 2.7B)	24.92	3957/15530	25.48	0/989	0.00	0/12709	0.00

Table 4.28: *Verifiability, generalisability and embedding error of the baseline and the robustly (adversarially) trained DNNs on the RUAR and the Medical datasets, for* $\mathbb{H}_{pert^\diamondsuit}$ *(*N_{base} *and* $N_{pert^\diamondsuit}$*) and* $\mathbb{H}_{pert}$ *(*N_{pert}*); for Marabou verifier.*

Table 4.28 shows the embedding error of our models and semantic subspaces. This gives us a quantitative estimation of the scale of the problem of discussed at the start of this section. Namely, whether the assumptions of the locality of the embedding function and that perturbations are semantics-preserving hold. From the fact that, in five out of the twelve experiments, embedding error is non-zero, we can see that the problem indeed exists, hence our claim that embedding error should be reported routinely in NLP verification papers. At the same time, the problem does not appear to be as severe as

one might fear: we note that at most 0.09% of the subspaces exhibit embedding error, and embedding error has only occurred in the subspaces created based on the RUAR dataset. Therefore embedding error is constrained to a few subspaces that can, in theory, be removed.

Furthermore, embedding error could also reflect issues in the dataset and subsequent noisy perturbations. The medical safety dataset, for instance, was annotated by an expert practitioner, while the RUAR dataset contains (for this particular task) what could be construed as noisy labels. For example *'are robot you you a'* is a sample that is found in the *negative* RUAR train set. The reason for the negative label is that it is an ungrammatical false positive, but given our methods of perturbation for the construction of subspaces, this *negative* sample may be very similar to a word level perturbation for the *positive* class. Concretely, for the model with the highest embedding error in Table 4.28 (i.e. N_{pert} s-bert 22M for RUAR dataset with 0.09% embedding error), some sentence pairs of *negative* samples with their accompanying perturbations contained in subspaces exhibiting embedding error are: *(Original: 'Are you a chump?', Perturbation: 'You a chump'), (Original: 'Are you a liar', Perturbation: 'You a liar'), (Original: 'if a computer can feel emotions, does that make you a computer or an actual human?', Perturbation: 'if a computer can feel, does that make it a machine or a person')*. Thus, the task of determining what queries require disclosure (e.g. should 'what is your favourite food' warrant disclosure?) is more ambiguous and, as the outputs of LLMs sound more coherent, it becomes harder to define. This area merits further research.

Embedding Error vs Generalisability and Verifiability

For comparison with the findings outlined in Section 4.3, we provide additional insights into verifiability and generalisability, also presented in Table 4.28. We first analyse the effect of cosine similarity filtering. Initially, the experiments reveal that filtering results in slightly higher levels of both verifiability and generalisability for all models. Given the conclusions in Section 4.3, the increase in verifiability is expected. However, the increase in generalisability is somewhat unexpected because, as demonstrated in Section 4.3, larger subspaces tend to exhibit greater generalisability, but filtering decreases the volume of the subspaces. Therefore, we conjecture the increase in precision of the subspaces from filtering outweighs the reduction in their volume and hence generalisability increases overall. The data therefore suggests that cosine similarity filtering can

serve as an additional heuristic for improving precision of the verified DNNs, and for further reducing the verifiability-generalisability gap. Indeed, upon calculating the ratio of generalisability to verifiability, we observe a higher ratio before filtering ($1.19 \rightarrow 0.99$ for RUAR and $0.99 \rightarrow 0.95$ for Medical). Recall that Section 4.3 already showed that our proposed usage of semantic subspaces can serve as a heuristic for closing the gap; and cosine similarity filtering provides opportunity for yet another heuristic improvement.

Moreover, the best performing model N_{pert} (s-bert 22M), results in $10,964$ (70.6%) medical perturbations and 9530 (21.65%) RUAR perturbations contained in the verified subspaces. While 21.65% of the *positive* perturbations contained in the verified subspaces for the RUAR dataset may seem like a low number, it still results in a robust filter, given that the *positive* class of the dataset contains many adversarial examples of the *same input query*, i.e. semantically similar but lexically different queries. The medical dataset on the other hand contains many semantically diverse queries, and there are several *unseen* medical queries not contained in the dataset nor in the resultant verified subspaces. However, given that the subspaces contain 70.6% of the *positive* perturbations of the medical safety dataset, an application of this could be to carefully curate a new dataset containing only queries with *critical* and *serious* risk-level labels defined by the World Economic Forum for chatbots in healthcare (see Section 4.2.1 and [184]). This dataset could be used to create verified filters centred around these queries to prevent generation of medical advice for these high-risk categories.

Overall, we find that **semantically-informed verification generalises well** across the different kinds of data to ensure guarantees on the output, and thus should aid in ensuring the safety of LLMs.

4.5 Conclusions and Future Work

Summary. This Chapter started with a general analysis of existing NLP verification approaches, with a view of identifying key components of a general NLP verification methodology. We then distilled these into a "NLP Verification Pipeline" consisting of the following six components:

1. dataset selection;
2. generation of perturbations;
3. choice of embedding functions;

4. definition of subspaces;

5. robust training;

6. verification via one of existing verification algorithms.

Based on this taxonomy, we make concrete selections for each component, and implement the pipeline using the tool ANTONIO [191]. ANTONIO allowed us to mix and match different choices for each pipeline component, enabling us to study the effects of varying the components of the pipeline in a algorithm-independent way. Our main focus was to identify weak or missing parts of the existing NLP verification methodologies. We proposed that NLP verification results should report, in addition to the standard verifiability metric, the following:

- whether they use geometric or semantic subspaces, and for which type of semantic perturbations;

- volumes, generalisability and embedding error of verified subspaces.

We finished the Chapter with a study of the current limitations of the NLP components of the pipeline and proposed possible improvements such as introducing a perturbations filter stage using cosine similarity. One of the major strengths of the pipeline is that each component can be improved individually.

Contributions. The major discoveries of this work were:

- In Section 4.3 we proposed generalisability as a novel metric, and showed that NLP verification methods exhibit a generalisability-verifiability trade-off. The effects of the trade-off can be severe, especially if the verified subspaces are generated naively (e.g. geometrically). We therefore strongly believe that generalisability should be routinely reported as part of NLP verification pipeline.

- In Sections 4.3 and 4.4 we showed that it is possible to overcome this trade-off by using several heuristic methods: defining semantic subspaces, training for semantic robustness, choosing a suitable embedding function and filtering with cosine similarity. All of these methods result in the definition of more precise verifiable subspaces; and all of them can be practically implemented as part of NLP verification pipelines in the future.

- In Section 4.4 we demonstrated that there are two key assumptions underlying the definition of subspaces that cannot be taken for granted. Firstly the LLMs, used as embedding functions, may not map semantically similar sentences to similar vectors in the embedding space. Secondly, our algorithmic methods for generating perturbations, whether by LLMs or otherwise, may not always be semantically-preserving operations. Both of these factors influence practical applications of the NLP verification pipeline.

- In Section 4.4 we demonstrated that even verified subspaces can exhibit semantic embedding errors: this effect is due to the tension between verification methods that are essentially geometric and the intuitively understood semantic meaning of sentences. By defining the *embedding error* metric and using it in our experiments, we demonstrated that the effects of embedding errors do not seem to be severe in practice; but this may vary from one scenario to another. It is important that NLP verification papers are aware of this pitfall, and report embedding error alongside verifiability and generalisability.

Finally, we claim as a contribution, a novel, coherent, methodological framework that allows us to include a broad spectrum of NLP, geometric, machine learning, and verification methods under a single umbrella. As illustrated throughout this Chapter, no previous publication in this domain covered this range and we believe that covering this broad range of methods is crucial for the development of this field.

Limitations and the Role of the Embedding Gap. In this work, we have shown the effect of the embedding gap in the NLP verification domain. In Section 4.3.1, we introduced the generalisability metric (Definition 17) to estimate how well subspaces capture semantically similar yet unseen sentences, providing a quantitative lens to examine this challenge. Sections 4.3.3 and 4.3.4 demonstrated that geometric subspaces struggle with the verifiability-generalisability trade-off, while semantic subspaces, constructed using semantic-preserving perturbations, show promise in mitigating the embedding gap by better aligning with data semantics.

The embedding gap is not unique to NLP, and manifests itself nearly in every domain where machine learning is applied. For example, in computer vision, the geometric definition of an ε-ball can include perturbations that no longer semantically resemble the original image (e.g., distortions that transform a given image into something unrecognisable). In the verification of neural network controllers, possible discrepancies arise

between interpretation of neural network inputs, such as velocity, distance, angle (i.e. have physical interpretation) and the way in which the neural network treats them as normalised input vectors [42, 199]. In NLP, the gap is amplified by the use of LLMs to map discrete sentences into continuous vector spaces. This process lacks a one-to-one correspondence between semantics and embeddings, exacerbating the challenge.

This problem is fundamental for machine learning methods deployed in NLP: they always rely on an "embedding function" that maps sentences into real vectors (on which machine learning algorithms operate). There is an implicit assumption that the embedding function works in a way that semantic similarity of sentences is reflected in geometric proximity of their embeddings. However, as general semantic similarity of sentences is not effectively computable, there is no hope that a perfect embedding function will ever be defined. Fundamentally, this is exactly the reason why machine learning (and not symbolic) approaches to NLP proved to be successful: they operate on the assumption that the embedding function is imperfect. As a consequence, any verification pipeline for NLP must include metrics that measure potential embedding errors. Section 4.4.6 is entirely devoted to defining this problem in mathematically precise terms and proposing an effective metric for measuring and reporting the severity and effects of the embedding errors.

This Chapter aims to quantify and address the embedding gap in the NLP domain. For better quantifying the effect of the embedding gap, we proposed precise metrics such as verifiability, generalisability and embedding error, and showed their interplay. This better understanding of the problem gave rise to our main positive result: the method that empirically reduces the effect of the embedding gap. While our findings mark progress, further research is needed to better align geometric representations with semantic meaning, especially in NLP contexts.

Future Work. Following from our in-depth analysis of the NLP perspective, we note that even if one has a satisfactory solution to all the issues discussed, there is still the problem of scalability of the available verification algorithms. For example, the most performant neural network verifier, $\alpha\beta$-Crown [36], can only handle networks in the range of tens of millions of trainable parameters. In contrast, in NLP systems, the base model of BERT [16] has around 110 million trainable parameters (considered small compared to modern LLMs – with trainable parameters in the billions!). It is clear that the rate at which DNN verifiers become more performant may never catch up with the rate

at which Large Language Models (LLMs) become larger. Then the question arises: how can this pipeline be implemented in the real world?

For future work, we propose to tackle this based on the idea of verifying a smaller DNN (classifier), manageable by verifiers, that can be placed upstream of a complex NLP system as a *safeguard*. We call this a *filter* (as mentioned in Section 4.2 and illustrated in Figure 4.1), and Figure 4.7 shows how a semantically informed verified filter can be prepended to an NLP system (here, an LLM) to check that safety-critical queries are handled responsibly, e.g. by redirecting the query to a tightly controlled rule-based system instead of a stochastic LLM. While there are different ways to implement the

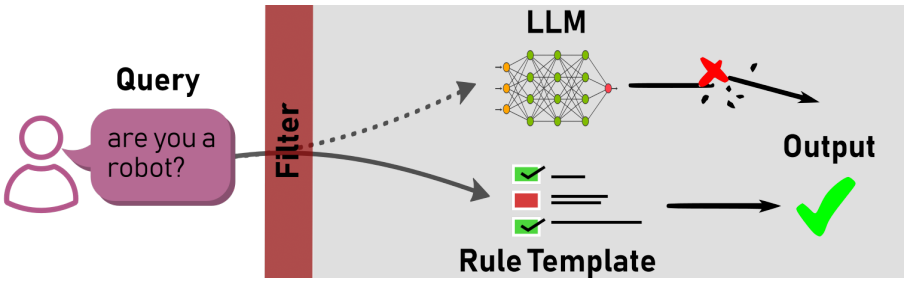


Figure 4.7: *In this figure, we show how a prepended, semantically informed verified filter added to an NLP system (here, an LLM), can check that safety-critical queries are handled responsibly, e.g. by redirecting the query to a tightly controlled rule-based system instead of a stochastic LLM.*

verification filters (e.g. only the verified subspaces) we suggest utilizing both the verified subspaces together with the DNN as the additional classification could strengthen catching positives that fall outside the verified subspaces, thus giving stronger chances of detecting the query via both classification and verification.

We note that the NLP community has recently proposed guardrails, in order to control the output of LLMs and create safer systems (such as from Open AI, NVIDIA and so on). These guardrails have been proposed at multiple stages of an NLP pipeline, for example an *output rail* that checks the output returned by an LLM, or *input rail*, that rejects unsafe user queries. In Figure 4.7, we show an application of our filter applied to the user input, which thus creates guarantees that a subset of safety critical queries are handled responsibly. In theory these verification techniques we propose may be applied to guardrails at different stages in the system, and we plan to explore this in future work.

A second future direction is to use this work to create NLP verification benchmarks. In 2020, the International Verification of Neural Networks Competition [32] (VNN-COMP) was established to facilitate comparison between existing approaches, bring researchers working on the DNN verification problem together, and help shape future direc-

tions of the field. However, for some time, the competition still lacked NLP verification benchmarks [200]. In 2024, we contributed a first NLP benchmark to VNN-COMP, using the methodology of this work, and the tool ANTONIO [5]. We intend to use this work for creating NLP verification benchmarks for future editions, to spread the awareness and attention to this field.

Chapter 5

Generalising the Training-Verification Pipeline to NIDS

Contents

5.1 Methodology 113

5.1.1 Feature Extraction and Input Representation 114

5.1.2 Feature Categorisation 115

5.1.3 Adapted Training-Verification Pipeline 116

5.2 Specifications 118

5.2.1 Global Specification Properties 119

5.2.2 Training on Global Properties 125

5.3 Experiments 127

5.4 Conclusions 132

In the preceding chapters, we developed a modular training-verification pipeline that aligns neural network properties defined over hyper-rectangles with both training objectives and formal verification. We applied this methodology to two domains: computer vision, where perturbations are geometric and continuous, and natural language processing, where robustness requires reasoning over discrete, semantically meaningful variations of sentences mapped into high-dimensional embeddings.

In this chapter, we extend the same framework to a fundamentally different domain and test whether it generalises to it: network intrusion detection systems (NIDS). Unlike computer vision and NLP, NIDS models operate on structured input vectors that combine both discrete and continuous features. Moreover, verification in this domain involves more than local perturbations—it requires satisfying high-level global properties, such as ensuring that specific traffic patterns are never misclassified.

In this chapter, we define and focus on global properties. Unlike the robustness properties discussed in Chapters 3 and 4, these properties are not limited to local perturbations around specific inputs. Instead, they require that all points within specified hyper-rectangles—encoding global logical constraints—are consistently classified into

the same class. These specifications are instances of *Classification Invariance* (Definition 14), which strictly generalises *Classification Robustness* from Chapter 3 beyond local neighbourhoods. While Chapter 3 highlighted limitations of CR (e.g., weak theoretical guarantees and ambiguity near decision boundaries) in certain settings, the hyper-rectangle formulation makes *Classification Invariance* a natural and practical choice here. By contrast, properties such as Standard Robustness or Lipschitz Robustness rely on local input–output relations around a specific input and are ill-suited to express global constraints spanning protocol and timing regimes. Adopting CI thus aligns the specification with the geometry of the domain while retaining a direct path to both training and verification.

We aim to derive a domain-specific pipeline for NIDS by adapting the core components of our methodology: defining verifiable subspaces, incorporating properties into training, and applying automated verification tools to provide formal guarantees. We show that only minimal domain-specific changes are needed—supporting the thesis that hyper-rectangles verification can be addressed via a unified, property-driven pipeline across diverse domains.

This chapter is structured as follows:

- Section 5.1 describes our methodology, including the structure of the NIDS input space and how we instantiate the pipeline.
- Section 5.2 formalises the different robustness properties that we consider.
- Section 5.3 presents experimental results on verification performance across a range of properties.
- We conclude in Section 5.4 by reflecting on the generalisability of the pipeline and domain-specific lessons learned.

5.1 Methodology

In this section, we adapt the training-verification pipeline introduced in Chapter 2 for NLP to the domain of NIDS. Our goal is to demonstrate that the same principled alignment of training and verification objectives applies in this new context, despite significant differences in data representation and threat models.

Unlike the NLP framework, where input data is embedded in high-dimensional semantic spaces, the NIDS framework involves structured network flow data composed of engineered features. These features are a mix of continuous and discrete values, often derived from low-level packet data or aggregated flow statistics.

5.1.1 Feature Extraction and Input Representation

Following [49], we use a restricted set of features to reduce over-fitting to spurious correlations and increase interpretability. Given a flow consisting of F packets, we extract features from the first f packets in the flow (Table 5.1). The packet-level features include:

- *Packet size*
- *Inter-arrival times (IATs)*
- *Packet flags*
- *Packet direction* — incoming or outgoing

We also include two flow-level features that are constant across all packets:

- *Transport protocol* — TCP or UDP
- *TimeElapsed* — the time since the last flow with the same Source/Destination IPs

Feature	Type	Role
`Time elapsed`	Continuous	Time since the last flow with the same Source/Destination IPs
`Transport protocol`	Discrete	TCP or UDP
`Packet direction`	Discrete	Incoming or outgoing
`Packet flags`	Discrete	Connection state indicator
`IATs`	Continuous	Inter-arrival times between the packet and the next one
`Packet size`	Continuous	Size of the packet

Table 5.1: *Categorisation of key NIDS features used in the training-verification pipeline.*

If a flow has fewer than f packets ($F < f$), we apply zero-padding to the feature slots corresponding to the missing packets. The input dimension m depends on the number of packets f and it is equivalent to two (transport protocol and time elapsed) plus four times (packets size, IATs, packet flags and packet direction) the number of packets: $m = 2 + 4 \cdot f$. We denote the feature value of the i-th packet in a flow $x \in \mathbb{R}^m$ as $x_{\textit{FeatureName-}i}$. This structured format allows us to explicitly define subspaces of the input space for verification by bounding or constraining feature values directly.

5.1.2 Feature Categorisation

Unlike feature attribution methods such as LIME [201] and SHAP [202], which identify features most influential to a model's decision, we take a verification-centric perspective: we specify which features should be constrained and in what way, allowing us to encode domain assumptions directly into robustness or global properties.

We classify features into three categories based on their interpretability by domain experts:

- *Related features* — directly relevant to the classification decision and expected to have a defined behavioural region;
- *Bounded features* — expected to vary within small intervals without affecting the output;
- *Unknown features* — whose impact on classification is unclear.

Consider the task of detecting Denial of Service (DoS) attacks. A feature like `TimeElapsed` (time since last flow between two endpoints) might be *related*, as short intervals could indicate flooding. Conversely, minor variations in packet sizes may not matter unless they exceed expected norms—making them *bounded*. Meanwhile, packet flags may be harder to interpret, so they are treated as *unknown*.

Definition 19 (Related Features) *Given a vector x, for a dimension i, we define a global interval* $[\alpha, \beta]$ *within which a classifier N must predict a fixed label c.*

$$\forall x \in \mathbb{R}^m \,.\, x_i \in [\alpha, \beta] \implies \arg\max N(x) = c$$

In other words, for all flows x where x_i falls within the interval $[\alpha, \beta]$, the model must classify the flow as c. In such cases, we will call i a related feature.

Definition 20 (Bounded Features) *For a dimension i, we require output consistency under small perturbations* $\gamma > 0$ *to an input* $\hat{x}$ *along i only:*

$$\forall x \in \mathbb{R}^m \,.\, [\forall j \neq i, x_j = \hat{x}_j] \wedge x_i \in [\hat{x}_i - \gamma, \hat{x}_i + \gamma] \implies \arg\max N(x) = c$$

Here, x is a full vector that equals $\hat{x}$ on all dimensions except i. This encodes local robustness to slight variation in feature i, which we call a bounded feature.

Both *related* and *bounded* features are special cases of *Classification Invariance* (Definition 14) over hyper-rectangles in which only the dimension i is allowed to vary. In the global (related) case, i ranges over $[\alpha, \beta]$ while other dimensions lie in their admissible domains. In the local (bounded) case, all dimensions match a reference flow $\hat{x}$ except i, which varies within a small interval around $\hat{x}_i$. In each case, the requirement is constant classification throughout the resulting hyper-rectangle.

Unknown Features. For unknown features, we do not explicitly constrain them during verification. The model is expected to learn any relevant correlations during training.

Global vs Local Context. Under our convention (Section 2.1.7), *local robustness* refers to CR on ε-balls $\mathbb{B}(\hat{x}, \varepsilon,)$ around a reference input. When we specialise *Classification Invariance* to single features, we use axis-aligned boxes in which only i varies: used *locally*, these boxes are centred at a reference $\hat{x}$; used *globally*, they are defined independently of any single input. Accordingly, in the global setting all constrained features—regardless of category—are treated as related features with bounds imposed directly over the input space.

5.1.3 Adapted Training-Verification Pipeline

We adapt the NLP pipeline to NIDS as shown in Figure 5.1. The pipeline consists of the following components:

1. **Given a network traffic dataset, introduce perturbations in traffic features.** We use the CIC IDS 2017 dataset [21] for training, extracting flows with an even split between benign and malicious samples. Malicious classes include *DoS Hulk*, *DoS Slowloris*, *FTP/SSH Bruteforce*, and *SQL injection* attacks. We evaluate performance on two test datasets: (i) CIC IDS 2018 [176], and (ii) a bespoke *DetGen* test set [203], which introduces variation in temporal (e.g., bandwidth throttling) and spatial (e.g., packet padding, webpage diversity) characteristics. We ensure proper separation between training and test data, following the guidelines of Flood et al. [177] to avoid data leakage and simulate detection of previously unseen attacks. We then introduce perturbations—both continuous (e.g., IATs or packet size) and discrete (e.g., transport protocol or packet flags)—that simulate plausible traffic variations.

2. **Pre-processing.** Each network flow is featurised to a fixed length by extracting the first f packets and zero-padding if needed. We compute packet-level features such as direction, size, inter-arrival time, and flags, alongside two flow-level features: transport protocol and `TimeElapsed`. The resulting feature vector has dimension $m = 2 + 4 \cdot f$, and is transformed and normalised into a structured input suitable for downstream learning and verification.

3. **Using domain knowledge and heuristics, define geometric subspaces.** Based on the feature categorisation from Section 5.1.2, we construct local ε-*balls* and global *hyper-rectangles*. Local subspaces are input-centred, capturing bounded perturbations around specific flows (e.g., small timing shifts). Global subspaces represent semantically significant regions (e.g., TCP flows with low IAT), allowing domain constraints to be directly encoded, similar to global specifications in systems like ACAS Xu [38].

4. **Use the geometric subspaces to train a classifier that is robust to label changes within the given subspaces.** We adapt PGD-based robust training to operate over the defined hyper-rectangles. All models share a fixed architecture: a fully connected feedforward network with three layers of sizes $(256, 128, 2)$, resulting in approximately 44,000 parameters and an input dimension of 42. This consistent setup enables fair comparison across properties and datasets.

5. **Verify the classifier's behaviour within the defined subspaces.** As in previous chapters, we use complete, sound methods to verify robustness. Specifically, we use the Marabou verifier [33], invoked through Vehicle [42], to verify that the model produces consistent outputs across both local and global perturbations.

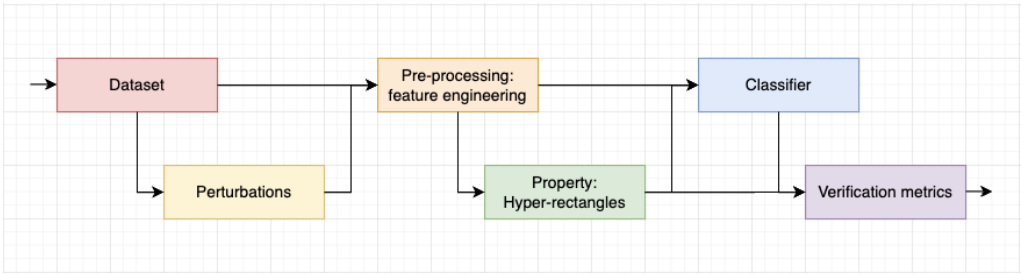


Figure 5.1: *Adapted training-verification pipeline for NIDS.*

5.2 Specifications

In adapting the training-verification pipeline to NIDS, the main challenge lies in expressing semantically meaningful constraints over a structured and partially discrete input space. While earlier chapters focused on local robustness properties, here we emphasise global specifications, which better align with NIDS semantics (e.g., ensuring all SYN flood traffic is rejected). These global properties describe large semantically coherent subspaces—rather than perturbation-based neighbourhoods—and express domain-level expectations about model behaviour.

Specification Framework. To express these properties in a rigorous and modular way, we use the Vehicle specification language [42], introduced in Section 2.1.6. Vehicle allows us to define logical constraints over feature vectors and verify them against a trained DNN using complete solvers such as Marabou [33].

Rather than working at the level of concrete input-output examples, each specification describes a family of flows satisfying a combination of logical and numeric constraints (e.g., "flows with valid TCP handshakes and small packet sizes"), and then asserts that all such flows must be assigned a specific label. This generalises the idea of classification robustness to the global setting, as discussed at the beginning of Chapter 5.

Specification Strategy. We design eight global properties representing normal or attack traffic patterns. Each property defines a subspace of the input space using combinations of predicates such as:

- `validInput` — basic validity checks;
- `validTCPHandshake`, `validHTTPGetRequest` — protocol structure;
- `validTimeElapsed`, `boundFlowDuration`, `validIATs` — temporal/spatial constraints;
- `validSizes`, `validSSHStart`, `validSSHTimeElapsed` — domain-specific features.

A complete description of all eight global specifications, along with snippets from the Vehicle code and an explanation of their structure, is provided in the next subsection.

On Local Properties. While our focus is on global properties, we also implement a simple local robustness experiment for comparison: we evaluate CR on ε-balls around specific flows. For alignment with our feature categories, we consider small boxes where only bounded features vary while related ones remain fixed. This confirms the applicability of local methods but also highlights their limited relevance for capturing broader NIDS behaviour.

5.2.1 Global Specification Properties

We formalise eight global properties for NIDS, each capturing a specific expected behaviour in network traffic—either legitimate or attack-related. These are encoded using the Vehicle specification language, which allows us to declaratively define complex subspaces using logical combinations of feature constraints. All global properties share a common structure: they define a logical predicate over the input space (e.g., a valid HTTP request, abnormal inter-arrival timing), and require the model to output a specific label (either benign or malicious) for all points in that subspace.

We categorise these properties into two types, depending on the expected classification output:

- **Positive properties**: regions of the input space that should always be classified as benign;
- **Negative properties**: regions that should always be flagged as malicious.

Before presenting the full properties, we define reusable components and constraints that serve as building blocks.

Constraint Building Blocks

We denote a flow as a feature vector $x \in \mathbb{R}^m$, where $m = 42$ is the number of input features. These correspond to a sequence of 10 packets, and are structured as follows:

- x_0: `TimeElapsed` between flows;
- x_1: `Protocol` indicator;
- $x_{2:11}$: `PacketDirection` for each packet;
- $x_{12:21}$: `PacketFlags` for each packet;

- $x_{22:31}$: `PacketIAT` (inter-arrival time) for each packet;
- $x_{32:41}$: `PacketSize` for each packet.

Property 1 (Valid Input) *A flow x is considered well-formed if all features lie within valid bounds (between 0.0 and 1.0), the first packet has zero inter-arrival time, the packet directions are either 0.0 or 1.0 and flags are within permitted values. We define:*

$$\begin{aligned}
\texttt{ValidBoundaries}(x) &\coloneqq \forall j \in [0,41],\ 0.0 \leq x_j \leq 1.0 \\
\texttt{ValidDirections}(x) &\coloneqq \forall j \in [2,11],\ x_j = 0.0 \ \vee\ x_j = 1.0 \\
\texttt{ValidFlags}(x) &\coloneqq \forall j \in [12,21],\ x_j \in \left\{ \frac{k}{256} \mid k \in \mathbb{N} \cap [0,255] \right\} \\
\texttt{ValidIATs}(x) &\coloneqq x_{22} = 0.0 \\
\texttt{ValidInput}(x) &\coloneqq \texttt{ValidBoundaries}(x) \ \wedge\ \texttt{ValidDirections}(x) \ \wedge \\
&\qquad \texttt{ValidFlags}(x) \ \wedge\ \texttt{ValidIATs}(x)
\end{aligned}$$

Property 2 (Valid TCP Handshake) *We define a valid TCP handshake as a sequence of three packets with specific directions (where 0.0 indicates an outgoing packet and 1.0 an incoming one), flags, and packet sizes.*

$$\begin{aligned}
\texttt{TCPDirections}(x) &\coloneqq x_2 = 0.0 \ \wedge\ x_3 = 1.0 \ \wedge\ x_4 = 0.0 \\
\texttt{TCPFlags}(x) &\coloneqq x_{12} = 2/256 \ \wedge\ x_{13} = 18/256 \ \wedge\ x_{14} = 16/256 \\
\texttt{TCPSizes}(x) &\coloneqq x_{32} = 52/1000 \ \wedge\ x_{33} = 52/1000 \ \wedge\ x_{34} = 40/1000 \\
\texttt{ValidTCPHandshake}(x) &\coloneqq \texttt{TCPDirections}(x) \ \wedge\ \texttt{TCPFlags}(x) \ \wedge\ \texttt{TCPSizes}(x)
\end{aligned}$$

Property 3 (Valid HTTP Get Request) *We define predicates for valid HTTP get requests by constraining the first the fourth and fifth packets directions, flags and sizes:*

$$\begin{aligned}
\texttt{HTTPDirections}(x) &\coloneqq x_5 = 0.0 \ \wedge\ x_6 = 1.0 \\
\texttt{HTTPFlags}(x) &\coloneqq x_{15} = 24/256 \ \wedge\ x_{16} = 16/256 \\
\texttt{HTTPSizes}(x) &\coloneqq 100/1000 \leq x_{35} \leq 500/1000 \ \wedge\ x_{36} = 40/1000 \\
\texttt{ValidHTTPGetRequest}(x) &\coloneqq \texttt{HTTPDirections}(x) \ \wedge\ \texttt{HTTPFlags}(x) \ \wedge\ \texttt{HTTPSizes}(x)
\end{aligned}$$

Property 4 (Valid SSH Start) *We define valid SSH session initiation by constraining the*

first five packets directions and flags and the first seven packets sizes:

$$\begin{aligned}
\texttt{SSHDirections}(x) &:= x_2 = 0.0 \land x_3 = 1.0 \land x_4 = 0.0 \land x_5 = 0.0 \land x_6 = 1.0 \\
\texttt{SSHFlags}(x) &:= x_{12} = 2/256 \land x_{13} = 18/256 \land x_{14} = 16/256 \land \\
&\quad x_{15} = 24/256 \land x_{16} = 16/256 \\
\texttt{SSHSizes}(x) &:= x_{32} = 60/1000 \land x_{33} = 60/1000 \land x_{34} = 52/1000 \land \\
&\quad \land x_{36} = 52/1000 \land x_{38} = 52/1000 \land ((x_{35} = 76/1000 \land \\
&\quad x_{37} = 93/1000) \lor (x_{35} = 93/1000 \land x_{37} = 76/1000)) \\
\texttt{ValidSSHStart}(x) &:= \texttt{SSHDirections}(x) \land \texttt{SSHFlags}(x) \land \texttt{SSHSizes}(x)
\end{aligned}$$

Property 5 (Valid Packet Sizes) *We define predicates for valid packet sizes by constraining the size of each packet to be* $\geq 40/1000$ *and the sum of the last six packet's sizes to be* $\geq 400/1000$*:*

$$\begin{aligned}
\texttt{ValidSizes}(x) &:= \forall j \in [32,41],\ x_j \geq 40/1000 \land \\
&\quad \sum_{j=36}^{41} x_j \geq 400/1000
\end{aligned}$$

Property 6 (Valid Inter Arrival Times) *We define valid inter-arrival times (IATs) using individual and aggregate bounds over selected packet indices.*

$$\begin{aligned}
\texttt{ValidIATs}(x) &:= x_{24} \leq 0.05 \land x_{25} \leq 0.05 \land x_{26} \leq 0.05 \land \\
&\quad x_{30} \leq 0.05 \land x_{31} \leq 0.05 \\
\texttt{BoundFlowDuration}(x) &:= \forall j \in [22,31],\ 1 \cdot 10^{-7} \leq x_j \leq 5 \cdot 10^{-6} \land \\
&\quad 1 \cdot 10^{-5} \leq x_{24} + x_{27} + x_{30} \leq 1.2 \cdot 10^{-5} \\
\texttt{InvalidFlowDuration}(x) &:= \forall j \in [22,31],\ x_j \geq 0.1 \quad \land \\
&\quad x_{24} + x_{27} + x_{30} \geq 1.12
\end{aligned}$$

Property 7 (Time Elapsed Constraints) *We define timing constraints related to the* `TimeElapsed` *feature. These reflect different attack scenarios.*

$$\begin{aligned}
\texttt{ValidTimeElapsed}(x) &:= x_0 = 0.0 \lor x_0 \geq 1 \cdot 10^{-3} \\
\texttt{InvalidHulkTimeElapsed}(x) &:= 1 \cdot 10^{-18} \leq x_0 \leq 1 \cdot 10^{-6} \\
\texttt{InvalidSlowHTTPTimeElapsed}(x) &:= 1 \cdot 10^{-18} \leq x_0 \leq 1 \cdot 10^{-3} \\
\texttt{InvalidSSHTimeElapsed}(x) &:= 1 \cdot 10^{-10} \leq x_0 \leq 1 \cdot 10^{-1}
\end{aligned}$$

Property (alphabetical)	Brief role	Defined in
`BoundFlowDuration`	Bounds on per–packet IATs and on a selected IAT sum to keep flow duration within range	Property 6
`InvalidFlowDuration`	Large IATs / large IAT sum indicating invalid (overlong) flow duration	Property 6
`InvalidHulkTimeElapsed`	`TimeElapsed` range indicative of Hulk–style behaviour	Property 7
`InvalidSlowHTTPTimeElapsed`	`TimeElapsed` range indicative of SlowHTTP–style behaviour	Property 7
`InvalidSSHTimeElapsed`	`TimeElapsed` range indicative of suspicious SSH activity	Property 7
`ValidHTTPGetRequest`	HTTP GET shape: directions, flags, and sizes constraints	Property 3
`ValidIATs`	Per–packet IAT upper bounds on selected indices	Property 6
`ValidInput`	Well–formed input: values in $[0,1]$, directions 0 or 1, flags quantised, first IAT $= 0$	Property 1
`ValidSizes`	Packet size minima; minimum total for last six packets	Property 5
`ValidSSHStart`	SSH session start pattern: first 5 directions/flags and specific size pattern	Property 4
`ValidTCPHandshake`	TCP 3–way handshake: directions, flags, and sizes	Property 2
`ValidTimeElapsed`	`TimeElapsed` either 0 or $\geq 10^{-3}$	Property 7

Table 5.2: *Alphabetical index of constraint building properties with cross–references to their formal definitions.*

Formal Definition of Global Properties

We now define the eight global properties using the building blocks from Section 5.2.1. For navigation, the primitive constraints are indexed alphabetically in Table 5.2, the properties themselves are summarised in Table 5.3, and compact Vehicle encodings are given in Figure 5.2. Each property is presented both in mathematical form and via its corresponding Vehicle specification.

Property 8 (Benign Traffic (Positive)) *This property captures well-formed, non-malicious HTTP flows. It requires a valid TCP handshake, a well-formed HTTP GET request, normal timing, bounded flow duration, and reasonable packet sizes. Formally:*

$$\begin{aligned} \mathtt{BenignTraffic} \coloneqq \forall x.\ & \mathtt{ValidInput}(x) \wedge \mathtt{ValidTCPHandshake}(x) \wedge \\ & \mathtt{ValidHTTPGetRequest}(x) \wedge \mathtt{ValidTimeElapsed}(x) \wedge \\ & \mathtt{BoundFlowDuration}(x) \wedge \mathtt{ValidSizes}(x) \Rightarrow \\ & \mathtt{Label}(x) = \mathit{Positive} \end{aligned}$$

Property 9 (Invalid Traffic (Negative)) *This captures malformed flows, including incorrect TCP handshakes or broken HTTP requests, and ensures they are flagged as malicious. Formally:*

$$\begin{aligned} \mathtt{InvalidTraffic} \coloneqq \forall x.\ & \mathtt{ValidInput}(x) \wedge (\neg \mathtt{ValidTCPHandshake}(x) \vee \neg \mathtt{ValidHTTPGetRequest}(x)) \Rightarrow \\ & \mathtt{Label}(x) = \mathit{Negative} \end{aligned}$$

Property 10 (Hulk Attack (Negative)) *The Hulk DoS pattern is characterised by rapid, repetitive HTTP requests. This property detects those using valid TCP handshakes and HTTP GET requests but with abnormally short time gaps between flows. Formally:*

$$\begin{aligned}\texttt{HulkAttack} \coloneqq\ & \forall x.\ \texttt{ValidInput}(x) \land \texttt{ValidTCPHandshake}(x) \land \\ & \texttt{ValidHTTPGetRequest}(x) \land \texttt{InvalidHulkTimeElapsed}(x) \Rightarrow \\ & \texttt{Label}(x) = \textit{Negative}\end{aligned}$$

Property 11 (SYN Flood Attack (Negative)) *SYN flood attacks often include incomplete handshakes or highly compressed inter-arrival timings. This property flags such abnormal cases. Formally:*

$$\begin{aligned}\texttt{SYNFloodAttack} \coloneqq\ & \forall x.\ \texttt{ValidInput}(x) \land (\neg\texttt{ValidTCPHandshake}(x) \lor \neg\texttt{ValidIATs}(x)) \Rightarrow \\ & \texttt{Label}(x) = \textit{Negative}\end{aligned}$$

Property 12 (Slow HTTP Test (Negative)) *Slow DoS attacks such as Slowloris present unusually large IATs or malformed HTTP headers. This property flags such combinations. Formally:*

$$\begin{aligned}\texttt{SlowHTTPTest} \coloneqq\ & \forall x.\ \texttt{ValidInput}(x) \land \texttt{ValidTCPHandshake}(x) \land \\ & (\neg\texttt{ValidHTTPGetRequest}(x) \lor \neg\texttt{ValidIATs}(x)) \Rightarrow \\ & \texttt{Label}(x) = \textit{Negative}\end{aligned}$$

Property 13 (Slow IATs Attack (Negative)) *A variant of slow DoS that shows prolonged timing and abnormal flow durations. It keeps packet headers valid but timing abnormal. Formally:*

$$\begin{aligned}\texttt{SlowIATsAttack} \coloneqq\ & \forall x.\ \texttt{ValidInput}(x) \land \texttt{ValidTCPHandshake}(x) \land \texttt{ValidHTTPGetRequest}(x) \land \\ & \texttt{InvalidSlowHTTPTimeElapsed}(x) \land \texttt{InvalidFlowDuration}(x) \Rightarrow \\ & \texttt{Label}(x) = \textit{Negative}\end{aligned}$$

Property 14 (SSH Brute Force (Negative)) *This property captures SSH sessions that begin correctly but repeat too rapidly, indicating brute-force behaviour. Formally:*

$$\begin{aligned}\texttt{SSHBruteForce} \coloneqq\ & \forall x.\ \texttt{ValidInput}(x) \land \texttt{ValidSSHStart}(x) \land \texttt{InvalidSSHTimeElapsed}(x) \Rightarrow \\ & \texttt{Label}(x) = \textit{Negative}\end{aligned}$$

Property 15 (Good SSH Login (Positive)) *This final property covers benign SSH traf-*

fic: valid session initialisation followed by acceptable time gaps. Formally:

$$\mathit{GoodSSHLogin} \coloneqq \forall x.\ \mathit{ValidInput}(x) \land \mathit{ValidSSHStart}(x) \land \neg \mathit{InvalidSSHTimeElapsed}(x) \Rightarrow \mathit{Label}(x) = \mathit{Positive}$$

```
@property
benignTraffic : Bool
benignTraffic = forall x . validInput x
  and validTCPHandshake x
  and validHTTPGetRequest x
  and validTimeElapsed x
  and boundFlowDuration x
  and validSizes x =>
  advises x pos
```

(a) `benignTraffic`

```
@property
invalidTraffic : Bool
invalidTraffic = forall x . validInput x
  and (not validTCPHandshake x
  or not validHTTPGetRequest x) =>
  advises x neg
```

(b) `invalidTraffic`

```
@property
hulkAttack : Bool
hulkAttack = forall x . validInput x
  and validTCPHandshake x
  and validHTTPGetRequest x
  and invalidHulkTimeElapsed x =>
  advises x neg
```

(c) `hulkAttack`

```
@property
synFloodAttack : Bool
synFloodAttack = forall x . validInput x
  and (not validTCPHandshake x
  or not validIATs x) =>
  advises x neg
```

(d) `synFloodAttack`

```
@property
slowHTTPTest : Bool
slowHTTPTest = forall x . validInput x
  and validTCPHandshake x
  and (not validHTTPGetRequest x
  or not validIATs x) =>
  advises x neg
```

(e) `slowHTTPTest`

```
@property
slowIATsAttack : Bool
slowIATsAttack = forall x . validInput x
  and validTCPHandshake x
  and validHTTPGetRequest x
  and invalidSlowHTTPTimeElapsed x
  and invalidFlowDuration x =>
  advises x neg
```

(f) `slowIATsAttack`

```
@property
sshBruteForce : Bool
sshBruteForce = forall x . validInput x
  and validSSHStart x
  and invalidSSHTimeElapsed x =>
  advises x neg
```

(g) `sshBruteForce`

```
@property
goodSSHLogin : Bool
goodSSHLogin = forall x . validInput x
  and validSSHStart x
  and not invalidSSHTimeElapsed x =>
  advises x pos
```

(h) `goodSSHLogin`

Figure 5.2: *Vehicle encodings for the eight global properties.*

Property Name	Label	Description
Benign Traffic	Benign	Valid flows with normal timing and structure
Invalid Traffic	Malicious	Broken TCP handshakes or HTTP requests
Hulk Attack	Malicious	Fast repetitive requests (low `TimeElapsed`)
SYN Flood Attack	Malicious	Incomplete handshakes or short IATs
Slow HTTP Test	Malicious	Abnormally large IATs or malformed GETs
Slow IATs Attack	Malicious	Long delays and irregular flow duration
SSH Brute Force	Malicious	Frequent SSH attempts with short intervals
Good SSH Login	Benign	Normal SSH login timing

Table 5.3: *Summary of the eight global NIDS verification properties.*

5.2.2 Training on Global Properties

Training with global specifications requires sampling flows from predefined hyper-rectangular regions in the input space and enforcing output consistency across them. These regions are defined based on the same constraints used for formal verification, allowing us to tightly couple the training and verification objectives. However, unlike local robustness training, where perturbations are applied around individual data-points, global training requires domain-specific knowledge to define semantically meaningful subspaces. As a result, we restrict training to a subset of global properties where this definition is feasible and well-structured.

Trainable Properties. Among the eight global properties defined in Section 5.2.1, we select three for adversarial training:

- `BenignTraffic`
- `HulkAttack`
- `SlowIATsAttack`

These properties are chosen because they admit a small, finite set of well-defined hyper-rectangles, either manually constructed or algorithmically generated. The remaining five properties would require an impractical number of rectangles to exhaustively cover their semantic constraints.

BenignTraffic We instantiate `BenignTraffic` with two hyper-rectangles that differ only in `TimeElapsed`: (i) flows with `TimeElapsed` $= 0$ (initial packets), and (ii) flows with `TimeElapsed` $\in [10^{-3}, 1.0]$ (regular follow-up traffic). Packet *directions* and *flags* are fixed to valid benign values (i.e., specific point values), while *IATs* and selected *sizes* are constrained by small intervals reflecting typical variation (not fixed constants). This mixture of fixed values and tight intervals reflects domain knowledge: some header fields should be invariant, whereas timing and payload sizes legitimately vary within bounded ranges. Indices follow Section 5.2.1.

Hulk Attack. The Hulk attack region is defined as a single hyper-rectangle in which all features match benign traffic, except for the `TimeElapsed` feature. This is set to an extremely short interval $[10^{-18}, 10^{-6}]$, indicating rapid-fire repetitive requests. This

```
# With time elapsed = 0.0
goodHTTP1 =
    # x_0 = TimeElapsed, x_1 = Protocol
    [[0.0, 0.0], [0.0, 0.0],
    # x_{2..11} = PacketDirection
    [0.0, 0.0], [1.0, 1.0], [0.0, 0.0], [0.0, 0.0], [1.0, 1.0],
    [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0],
    # x_{12..21} = PacketFlags
    [2/256, 2/256], [18/256, 18/256], [16/256, 16/256], [24/256, 24/256], [16/256, 16/256],
    [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0],
    # x_{22..31} = PacketIAT
    [0.0, 1.0], [0.000001, 0.05], [0.000001, 0.05], [0.000001, 0.05], [0.000001, 0.05],
    [0.000001, 0.05], [0.000001, 0.05], [0.000001, 0.05], [0.000001, 0.05], [0.000001, 0.05],
    # x_{32..41} = PacketSize
    [52/1000, 52/1000], [52/1000, 52/1000], [40/1000, 40/1000], [100/1000, 500/1000], [40/1000, 40/1000],
    [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0]]

# With time elapsed between [0.001, 1.0]
goodHTTP2 =
    # x_0 = TimeElapsed, x_1 = Protocol
    [[0.001, 1.0], [0.0, 0.0],
    # x_{2..11} = PacketDirection
    [0.0, 0.0], [1.0, 1.0], [0.0, 0.0], [0.0, 0.0], [1.0, 1.0],
    [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0],
    # x_{12..21} = PacketFlags
    [2/256, 2/256], [18/256, 18/256], [16/256, 16/256], [24/256, 24/256], [16/256, 16/256],
    [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0],
    # x_{22..31} = PacketIAT
    [0.0, 1.0], [0.000001, 0.05], [0.000001, 0.05], [0.000001, 0.05], [0.000001, 0.05],
    [0.000001, 0.05], [0.000001, 0.05], [0.000001, 0.05], [0.000001, 0.05], [0.000001, 0.05],
    # x_{32..41} = PacketSize
    [52/1000, 52/1000], [52/1000, 52/1000], [40/1000, 40/1000], [100/1000, 500/1000], [40/1000, 40/1000],
    [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0]]
```

configuration captures the core semantics of the Hulk DoS pattern and is labelled as malicious.

```
hulk =
    # x_0 = TimeElapsed, x_1 = Protocol
    [[0.000000000000000001, 0.000001], [0.0, 0.0],
    # x_{2..11} = PacketDirection
    [0.0, 0.0], [1.0, 1.0], [0.0, 0.0], [0.0, 0.0], [1.0, 1.0],
    [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0],
    # x_{12..21} = PacketFlags
    [2/256, 2/256], [18/256, 18/256], [16/256, 16/256], [24/256, 24/256], [16/256, 16/256],
    [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0],
    # x_{22..31} = PacketIAT
    [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0],
    [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0],
    # x_{32..41} = PacketSize
    [52/1000, 52/1000], [52/1000, 52/1000], [40/1000, 40/1000], [100/1000, 500/1000], [40/1000, 40/1000],
    [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0]]
```

Slow IATs Attack. This property is defined by 10 separate hyper-rectangles, each targeting one of the packet inter-arrival times (IATs). For each rectangle, one IAT feature is constrained to be greater than 0.1, while others remain unrestricted. This setup simulates low-rate attacks characterised by delays between packets. All hyper-rectangles are labelled as malicious.

```
slowIATsAttacks =
    # x_0 = TimeElapsed, x_1 = Protocol
    [[0.0000000000000001, 0.001], [0.0, 0.0],
    # x_{2..11} = PacketDirection
    [0.0, 0.0], [1.0, 1.0], [0.0, 0.0], [0.0, 0.0], [1.0, 1.0],
    [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0],
    # x_{12..21} = PacketFlags
    [2/256, 2/256], [18/256, 18/256], [16/256, 16/256], [24/256, 24/256], [16/256, 16/256],
    [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0],
    # x_{22..31} = PacketIAT
    [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0],
    [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0],
    # x_{32..41} = PacketSize
    [52/1000, 52/1000], [52/1000, 52/1000], [40/1000, 40/1000], [100/1000, 500/1000], [40/1000, 40/1000],
    [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0]]

# Create 10 rectangles by modifying each IAT
for i in range(10):
    temp = copy.deepcopy(slowIATsAttacks)
    temp[i + 22][0] = 0.1  # Increase IAT lower bound
    hyperrectangles.append(temp)
```

Balanced Sampling. During training, we sample an equal number of points from each benign and malicious region. This balanced adversarial sampling ensures the classifier learns to associate subspace semantics with their corresponding labels, rather than overfitting to any one scenario. Sampling is repeated across all rectangles and flows to create a diverse training set with consistent constraints.

Limitations. Other global properties—such as the SYN flood, Slow HTTP Test, or SSH-related specifications—are more expressive but require disjunctions over many features or combinations of logical predicates that cannot be easily translated into a tractable number of hyper-rectangles. While they remain suitable for verification and generalisation experiments, their complexity renders them infeasible for inclusion in adversarial training without extensive domain engineering.

5.3 Experiments

We evaluate our training-verification pipeline on the NIDS domain using the global properties defined in Section 5.2.1. The experiments are designed to answer two core questions: (i) do verified global properties correlate with stronger empirical performance? and (ii) can training on a subset of properties improve robustness to adversarial attacks and generalisation?

Following the evaluation methodology from Chapter 3, we assess both formal verification (success rate) and empirical performance (accuracy, F1 score, CSec). Models are trained on CIC IDS 2017, and tested on CIC IDS 2018 and DetGen for cross-distribution

generalisation (see Section 5.1.3).

Cross-Dataset and Cross-Attack Generalisation

We begin by evaluating whether training for global properties improves generalisation across datasets and attack types. To this end, we compare three models: N_{base}, our benchmark model trained on standard DoS traffic from the CIC IDS 2017 dataset [21]; N_{pert}, a version trained by us using PGD-based adversarial training (Section 5.1.3) on the three trainable DoS properties defined in Section 5.2.2; and N_{LUCID}, a state-of-the-art DoS classifier from the literature (LUCID [167]).

Table 5.4 shows that training on a small number of well-defined global properties leads to substantial improvements in cross-dataset generalisation. Specifically, N_{pert} improves F1 score by approximately 0.35–0.4 when evaluated on DoS traffic from CIC IDS 2018 [176] and DetGen [203]. In contrast, the more complex N_{LUCID} model does not achieve similar gains, highlighting the effectiveness of targeted adversarial training.

This improvement also extends to cross-attack generalisation. When trained exclusively on DoS traffic, N_{pert} generalises to SSH Brute Force attacks from the CIC IDS 2018 and DetGen datasets with scores of 0.8871 and 0.8938, respectively. This suggests that our training process induces less overfit to specific attack classes. As our adversarial training generates synthetic flows aligned with our specifications, the model is exposed to a richer set of training examples. For instance, it learns to associate low `TimeElapsed` with malicious behaviour and high `TimeElapsed` with benign behaviour. Since SSH Brute Force traffic often exhibits low inter-flow arrival times, the model naturally generalises to this new context, despite having only seen DoS traffic during training.

Classifier	Test Data	F1 Score
Cross-dataset Generalisation		
N_{base}	DoS (CIC IDS 2018)	0.5583
N_{base}	DoS (DetGen)	0.5622
N_{LUCID}	DoS (CIC IDS 2018)	0.5421
N_{LUCID}	DoS (DetGen)	0.5468
$\boldsymbol{N_{pert}}$	**DoS (CIC IDS 2018)**	**0.9111**
$\boldsymbol{N_{pert}}$	**DoS (DetGen)**	**0.9521**
Cross-attack Generalisation		
N_{base}	SSH (CIC IDS 2018)	0.4456
N_{base}	FTP (DetGen)	0.4914
$\boldsymbol{N_{pert}}$	**SSH (CIC IDS 2018)**	**0.8871**
$\boldsymbol{N_{pert}}$	**SSH (DetGen)**	**0.8059**

Table 5.4: *F1 scores for NIDS models across datasets and attack types.*

Verification of Global Properties

We now verify the three models using the formal specifications defined in Section 5.2.1. Since the specifications are global rather than local, we do not express verifiability as a percentage of verified subspaces. Instead, the success of the global specifications is presented as a binary outcome: either verified or not verified.

Table 5.5 summarises the verification results, obtained using the Vehicle framework. As expected, neither N_{base} nor N_{LUCID} verify any property. In contrast, our adversarially trained N_{pert} model successfully verifies five of the eight global specifications.

Notably, N_{pert} verifies the three properties it was trained on: `BenignTraffic`, `HulkAttack`, and `SlowIATsAttack`. It also generalises well enough to verify two additional properties—`SSHBruteForce` and `GoodSSHLogin`—which are structurally similar to the first two. The key difference lies in the protocol used (SSH vs HTTP), further supporting the idea that semantically aligned training improves verification beyond the original targets.

Moreover, our specifications allow precise interpretation of feature interactions. For instance, the `HulkAttacks` and `SlowIATsAttacks` specifications both require the `TimeElapsed` feature to be less than some bound, β_{Hulk} and $\beta_{SlowIAT}$, respectively, whilst the `SlowIATsAttacks` specification additionally constrains the IATs of a flow to be large. Maximising these β bounds, we find that this extra IAT constraint allows us to verify `SlowIATsAttacks` specifications when β_{Hulk} is several orders of magnitude smaller than $\beta_{SlowIAT}$ (in our case $\beta_{Hulk} = 0.000001$ and $\beta_{SlowIAT} = 0.1$). This demonstrates that unusually high IATs contribute towards classifying flows as malicious.

Classifier	**Property**							
	Benign Traffic	**Invalid Traffic**	**Hulk**	**SYN Flood**	**Slow HTTP**	**Slow IATs**	**SSH Brute Force**	**Good SSH Login**
N_{base}	×	×	×	×	×	×	×	×
N_{LUCID}	×	×	×	×	×	×	×	×
$\mathbf{N_{pert}}$	√	×	√	×	×	√	√	√

Table 5.5: *Verifiability of the networks over our eight properties.*

Comparison with Local Robustness

We compare our approach against PGD-based adversarial training with $\varepsilon = 0.1$. Here, the local property enforced is *Classification Robustness (CR)*: for each reference flow $\hat{x}$, the predicted label remains constant throughout the ε-ball $\mathbb{B}(\hat{x}, \varepsilon)$. Although we can train a model that *verifiably* satisfies CR around each reference input, this yields no improve-

ment in cross-dataset generalisation: the locally robust model attains an F1 of only 0.5792 on the test set. Intuitively, these *local perturbations*—i.e., ℓ_p-bounded changes within $\mathbb{B}(\hat{x}, \varepsilon)$—do not respect the structure of network flows (protocol logic, header semantics, long-range timing), so the resulting adversarial examples convey little information about out-of-distribution traffic. In contrast, our *global* hyper-rectangle bounds encode domain-level constraints (e.g., timing regimes, protocol patterns), which better transfer across datasets and attack families.

Scalability: Specification Transferability

To evaluate the scalability of our approach, we transfer verified specifications from a small model to larger architectures (up to 2.8M parameters) without retraining bounds. This is a similar scale to models in verification competitions [32].

Parameters	Time (s)	Verifiable?
43,776	41	✓
186,498	47	✓
449,154	239	✓
963,330	1023	✓
2,842,882	19708	✓

Table 5.6: *Verification of the* `BenignTraffic` *specification across model sizes.*

All models verified successfully using the original specification and training bounds, showing that verified subspaces are transferable across architectures. Verification time increases with model size (Table 5.6), but remains feasible for models up to several million parameters.

Adversarial Robustness under Local Attacks

We test the Constraint Security (Definition 11) of globally trained models against unconstrained local attacks, using the FGSM and Momentum Iterative methods. Figure 5.3 shows that training with global properties enhances CSec, especially under FGSM perturbations.

While performance under MI-FGSM is more variable, models with higher training accuracy on specification-based adversarial data tend to exhibit stronger CSec overall. This suggests that global training confers partial robustness even under local perturbations, despite not being directly optimised for them.

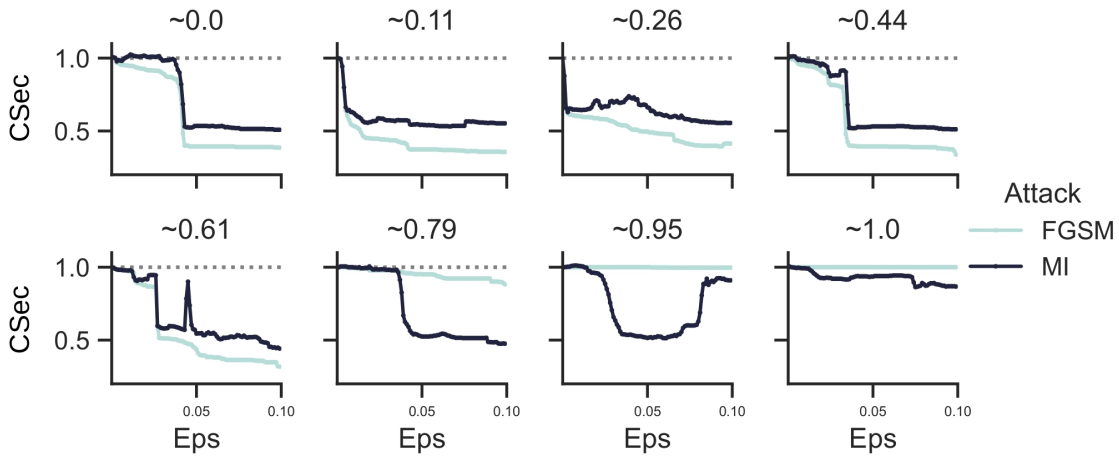


Figure 5.3: *Constraint Security (CSec) under local adversarial attacks, plotted against perturbation size ε.*

Coverage and Validity of Specifications

In most verification literature, specifications are assumed to be correct by construction. In contrast, since our global properties are manually engineered, we go a step further and validate their correctness empirically. To do this, we compute their coverage over labelled flows in the training and test datasets—that is, we calculate the proportion of dataset flows that satisfy the constraints of each specification. This mirrors the generalisability metric from Definition 17, but instead of evaluating on unseen inputs, we evaluate against labelled flows from the datasets themselves.

Table 5.7 summarises the results. Notably, the complete specifications for malicious traffic match nearly all ground truth flows (≈100%), confirming their alignment with the label semantics. Even the smaller, verifiable subsets of these specifications achieve high coverage—ranging from 40.94% to 82.83%—highlighting that they capture large and meaningful portions of the data. In contrast, benign specifications achieve lower coverage (from 0.12% to 10.88%), which is expected, as they are designed to target narrow traffic categories such as HTTP or SSH traffic. Given that these represent a small fraction of overall benign flows, this lower coverage still provides a valuable result. Overall, the results confirm that our specifications are valid, semantically meaningful, and grounded in realistic traffic patterns.

Attack	Type	Dataset	Coverage (#)	Coverage (%)
DoS	Benign	Train	13,488 / 189,796	7.11%
		Test	4,696 / 211,174	2.22%
	Malicious	Train	190,507 / 190,663	99.92%
		Test	221,801 / 221,801	100.00%
	Malicious*	Train	125,087 / 190,663	65.61%
		Test	183,727 / 221,801	82.83%
SSH	Benign	Train	748 / 6,876	10.88%
		Test	123 / 102,624	0.12%
	Malicious	Train	6,964 / 6,964	100.00%
		Test	108,639 / 108,639	100.00%
	Malicious*	Train	2,851 / 6,964	40.94%
		Test	62,402 / 108,639	57.44%

Table 5.7: *Coverage of specifications over labelled dataset flows.* ***Malicious**** *denotes flows matched by the verified subset of the full malicious specifications.*

5.4 Conclusions

In this chapter, we presented the first systematic application of neural network verification to network intrusion detection systems. Building on formally specified global properties, we demonstrated how verification can serve not only as a post-hoc check, but as a guiding principle for model design and training. We developed a verification-guided training pipeline that combines specification-based adversarial training with formal guarantees over meaningful regions of the input space. Our models generalise across datasets and attack classes, achieving significant gains in accuracy and robustness, while maintaining verifiability. By constraining training to high-confidence regions, we improved detection of subtle attacks (e.g., low-rate DoS, SSH brute force) and reduced over-fitting to dataset-specific artefacts. The use of a high-level specification language (Vehicle) enabled human-readable formal properties and facilitated the design of structured training procedures. We further showed that specifications can transfer across models of varying size and complexity, and that robustness to global properties also improves resistance to unconstrained local perturbations. Overall, this chapter illustrates that integrating verification into the NIDS pipeline enhances both performance and trustworthiness. Future work could explore richer specification languages, local property synthesis, or automated specification discovery for broader attack classes.

Chapter 6

Conclusions

Contents

6.1 Summary of Contributions 133

6.2 Cross-Domain Insights 134

6.3 Limitations 135

6.4 Guidelines for Practitioners 136

6.5 Future Work 136

6.6 Concluding Remarks 137

6.1 Summary of Contributions

This thesis examined how to *specify*, *train for*, and *verify* robustness properties of DNNs in a unified way across computer vision, NLP, and NIDS. We studied two complementary classes of specifications:

- **Local properties** that constrain behaviour on ε-balls $\mathbb{B}(\hat{x}, \varepsilon)$ around a reference input (i.e., *Classification Robustness* on local regions).
- **Global properties** that enforce *Classification Invariance* over axis-aligned hyper-rectangles (e.g., protocol and timing-based constraints for network flows).

A central theme is *verifiability*: the extent to which a meaningful property can be *defined* precisely and *checked*. In this thesis we emphasise that verifiability depends primarily on the specification itself: whether the property is formulated in a representation that faithfully captures its semantic intent (for example, using hyper-rectangles to encode local or global regions) and can be applied consistently across training and verification.

Chapter 3 (Computer Vision). We identified and compared four robustness properties [1]—*classification robustness (CR)*, *strong classification robustness (SCR)*, *standard*

[1] Our treatment of robustness notions (CR/SCR/SR/LR) is not exhaustive; we focus on the formulations most central to the verification pipelines we study.

robustness (SR), and *Lipschitz robustness (LR)*—highlighting logical relations (e.g., LR implies SR, while SR and SCR are incomparable). We demonstrated empirically that mismatches between the training objective and the verified property degrade guarantees, and we proposed a shared evaluation triad (constraint satisfaction, security, and accuracy) to make guarantees comparable across properties.

Chapter 4 (NLP). To address the *embedding gap*, we introduced *semantic subspaces*: for each sentence, we collect paraphrases, embed them, and enclose their point set in a hyper-rectangle. We trained models on these subspaces and verified behaviour directly in the continuous space, guided by semantic information. This made properties both *definable* (in terms of geometric bounds) and *checkable* (by complete verifiers), despite embeddings being non-invertible.

Chapter 5 (NIDS). We defined eight *global* properties that capture protocol logic, timing regimes, and packet-structure constraints. Expressed as hyper-rectangles and Boolean combinations of primitives, they support both automated verification and *property-driven training*. Our results show that, despite differences in data type and semantics, the same training–verification methodology applies. Importantly, we demonstrate how this approach generalises to *global* properties—a setting not addressed in the vision or NLP chapters—and that the modular construction scales across domains with minimal adjustments.

6.2 Cross-Domain Insights

Property Alignment as First-Class Design Choice. Across domains, guarantees improved most when the *training objective matched the property being verified*. Training for CR and verifying CR is more effective than training for SR and verifying CR, and similarly for global NIDS constraints. This argues for a *specification-first* workflow: select the property that captures intent, then shape data augmentation, loss design, and verification around it.

Subspace Construction Determines Verifiability. Geometric regions that align with semantics are easier to verify. In vision, simple ε-balls often suffice; in NLP, semantic subspaces offer a superior verifiability–generalisability trade-off to raw ε-balls; in NIDS,

rule-driven hyper-rectangles compactly encode domain knowledge. When the chosen region shape does not reflect the task semantics, verification degrades: bounds become loose, case splits explode, or the certified region includes inputs that are semantically implausible.

Managing the Embedding Gap. Embedding choices (normalisation, hand-crafted encodings, learned sentence transformers) reshape input geometry. Where semantics and geometry diverge, *verifiability* suffers. Our results show that semantics-aware subspaces and specifications mitigate this gap in practice.

A Practical Evaluation Suite. We advocated three complementary metrics: *constraint satisfaction* (per-input certification on local ε-balls, Definition 10), *constraint security* (resistance to attacks within certified regions), and *constraint accuracy* (task accuracy consistent with the constraint). In addition, we report *verifiability* (Definition 16), a per-region metric that counts certified regions regardless of shape (ε-balls or hyper-rectangles). If we take the regions in Definition 16 to be the same ε-balls used for CSat, the verifiability score equals CSat. Under ℓ_∞, ε-balls coincide with ε-cubes, and verifiability then *strictly generalises* CSat by also accommodating arbitrary axis-aligned hyper-rectangles. Finally, we use *generalisability* (Definition 17) to estimate the coverage of semantically similar unseen sentences by a set of regions; in NIDS we complement this with cross-dataset F1 as an external generalisation check. Taken together, these measures support consistent comparison across domains and specification styles.

6.3 Limitations

While the methodology is general, it carries assumptions:

- **Specification coverage.** Global properties are necessarily partial views of domain behaviour; writing complete rule sets is labour-intensive and may miss rare but important cases.
- **Embedding dependence.** NLP guarantees are conditioned on the embedding function; different encoders yield different subspaces and verification difficulty.
- **Verifier scalability.** Complete solvers (e.g., SMT-based) improve precision but can struggle at scale; abstract-interpretation methods scale but may under-certify due

to loose over-approximations.

- **Subspace shape.** Axis-aligned hyper-rectangles offer simplicity and tool support, but richer shapes (zonotopes, polytopes) could capture semantics more tightly at the cost of solver complexity.
- **Scope of properties.** The properties studied are primarily invariance-style specifications (local invariance and global rule satisfaction). Extending the methodology to alternative specification styles remains open.

6.4 Guidelines for Practitioners

Rather than prescribing a checklist, we recommend a specification-first workflow. Begin by stating the property in a form that can be encoded and reused. Shape subspaces so that they reflect domain semantics (e.g., hyper-rectangles over interpretable features, semantic subspaces for paraphrases). Align the training objective with the property to be certified, minimising training–verification mismatch. Finally, accompany results with property-aware evaluation—constraint satisfaction, security, and accuracy—to communicate guarantees precisely.

6.5 Future Work

Robustness properties in vision. Extend the analysis of CR/SCR/SR/LR by studying training objectives that target stronger properties directly, and by developing other robustness metrics that detect and mitigate training–verification mismatch.

Semantic subspaces for NLP. Improve the construction of paraphrase-based subspaces (e.g., better paraphrase mining and filtering), study the effect of different encoders on verifiability, and explore specification styles beyond label invariance while remaining geometry-compatible.

Global properties for NIDS. Enrich the property set with temporal and stateful predicates (e.g., across longer flow windows), automate rule discovery from logs, and integrate property-driven training and verification into deployment pipelines to support continuous verification.

Region classes and tooling. Investigate region families beyond axis-aligned boxes when they provide tighter semantic fit, and develop lightweight tooling to express, reuse, and test properties across domains.

6.6 Concluding Remarks

Reliable machine learning begins with precise, verifiable specifications rather than solely optimising for benchmark accuracy. It starts with *writing the right property*, constructing *verification-friendly subspaces* that reflect domain semantics, and *training for exactly that property*. This thesis showed that such a specification-first, cross-domain approach is practical and beneficial. We hope these principles help make certified behaviour a routine part of modern ML practice.

References

[1] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and Harnessing Adversarial Examples*. 2015. arXiv: 1412.6572 [stat.ML].

[2] Christian Szegedy et al. *Intriguing properties of neural networks*. 2014. arXiv: 1312.6199 [cs.CV].

[3] Patrick Cousot and Radhia Cousot. "Abstract interpretation: past, present and future". In: *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 2014, pp. 1–10.

[4] Shixiang Gu and Luca Rigazio. "Towards deep neural network architectures robust to adversarial examples". In: *arXiv preprint arXiv:1412.5068* (2014).

[5] Christopher Brix et al. "The Fifth International Verification of Neural Networks Competition (VNN-COMP 2024): Summary and Results". In: *arXiv preprint arXiv:2412.19985* (2024).

[6] Kai Kugler et al. "InvBERT: Reconstructing Text from Contextualized Word Embeddings by inverting the BERT pipeline". In: *arXiv preprint arXiv:2109.10104* (2021).

[7] Johannes Stallkamp et al. "The German Traffic Sign Recognition Benchmark: A multi-class classification competition". In: *IEEE International Joint Conference on Neural Networks*. 2011, pp. 1453–1460.

[8] Ekaterina Komendantskaya, Wen Kokke, and Daniel Kienitz. "Continuous Verification of Machine Learning: a Declarative Programming Approach". In: *PPDP '20: 22nd International Symposium on Principles and Practice of Declarative Programming, Bologna, Italy, 9-10 September, 2020*. ACM, 2020, 1:1–1:3.

[9] Aleksander Madry et al. "Towards Deep Learning Models Resistant to Adversarial Attacks". In: *International Conference on Learning Representations*. 2018.

[10] Thomas Flinkow et al. "A Generalised Framework for Property-Driven Machine Learning". In: *arXiv preprint arXiv:2505.00466* (2025).

[11] Thomas Flinkow, Barak A Pearlmutter, and Rosemary Monahan. "Comparing differentiable logics for learning with logical constraints". In: *Science of Computer Programming* 244 (2025), p. 103280.

[12] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814.

[13] Shun-ichi Amari. "Backpropagation and stochastic gradient descent method". In: *Neurocomputing* 5.4-5 (1993), pp. 185–196.

[14] Diederik P Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *ICLR (Poster)*. 2015.

[15] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. 2017. arXiv: 1708.07747 [cs.LG].

[16] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. DOI: 10.48550/ARXIV.1810.04805. URL: https://arxiv.org/abs/1810.04805.

[17] Nils Reimers and Iryna Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 3982–3992. DOI: 10.18653/v1/D19-1410. URL: https://aclanthology.org/D19-1410.

[18] Niklas Muennighoff. *SGPT: GPT Sentence Embeddings for Semantic Search*. 2022. arXiv: 2202.08904 [cs.CL].

[19] David Gros, Yu Li, and Zhou Yu. "The RUA-Robot Dataset: Helping Avoid Chatbot Deception by Detecting User Questions About Human or Non-Human Identity". In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 2021, pp. 6999–7013.

[20] Gavin Abercrombie and Verena Rieser. “Risk-graded Safety for Handling Medical Queries in Conversational AI”. In: *Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing*. 2022, pp. 234–243.

[21] Gints Engelen, Vera Rimmer, and Wouter Joosen. “Troubleshooting an Intrusion Detection Dataset: the CICIDS2017 Case Study”. In: *2021 IEEE Security and Privacy Workshops (SPW)*. IEEE. 2021, pp. 7–12.

[22] Iman Sharafaldin, Arash Habibi Lashkari, Ali A Ghorbani, et al. “Toward generating a new intrusion detection dataset and intrusion traffic characterization.” In: *ICISSp* 1 (2018), pp. 108–116.

[23] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. “Adversarial machine learning at scale”. In: *arXiv preprint arXiv:1611.01236* (2016).

[24] Nicolas Papernot et al. *Crafting Adversarial Input Sequences for Recurrent Neural Networks*. 2016. arXiv: 1604.08275 [cs.CR].

[25] Connor Shorten and Taghi M. Khoshgoftaar. “A survey on Image Data Augmentation for Deep Learning”. In: *J. Big Data* 6 (2019), p. 60.

[26] Radu Balan, Maneesh Singh, and Dongmian Zou. “Lipschitz properties for deep convolutional networks”. In: *Contemporary Mathematics* 706 (2018), pp. 129–151.

[27] Patricia Pauli et al. “Training robust neural networks using Lipschitz bounds”. In: *IEEE Control Systems Letters* (2021).

[28] Henry Gouk et al. “Regularisation of neural networks by enforcing Lipschitz continuity”. In: *Machine Learning* 110.2 (2021), pp. 393–416.

[29] Jingyi Xu et al. “A Semantic Loss Function for Deep Learning with Symbolic Knowledge”. In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by Jennifer G. Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 5498–5507.

[30] Marc Fischer et al. “DL2: Training and Querying Neural Networks with Logic”. In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 1931–1941. URL: http://proceedings.mlr.press/v97/fischer19a.html.

[31] Matthias Althoff. “An Introduction to CORA 2015”. In: *Proc. of the 1st and 2nd Workshop on Applied Verification for Continuous and Hybrid Systems*. EasyChair, Dec. 2015, pp. 120–151. DOI: 10.29007/zbkv. URL: https://easychair.org/publications/paper/xMm.

[32] Christopher Brix et al. “First three years of the international verification of neural networks competition (VNN-COMP)”. In: *International Journal on Software Tools for Technology Transfer* 25.3 (2023), pp. 329–339.

[33] Haoze Wu et al. *Marabou 2.0: A Versatile Formal Analyzer of Neural Networks*. 2024. arXiv: 2401.14461 [cs.AI].

[34] Gagandeep Singh et al. “An abstract domain for certifying neural networks”. In: *Proceedings of the ACM on Programming Languages* 3.POPL (2019), pp. 1–30.

[35] Kaidi Xu et al. “Fast and Complete: Enabling Complete Neural Network Verification with Rapid and Massively Parallel Incomplete Verifiers”. In: *International Conference on Learning Representation (ICLR)*. 2021.

[36] Shiqi Wang et al. “Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 29909–29921.

[37] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. “The satisfiability modulo theories library (SMT-LIB). www”. In: *SMT-LIB. org* 2 (2016), p. 68.

[38] Guy Katz et al. “Reluplex: An efficient SMT solver for verifying deep neural networks”. In: *International conference on computer aided verification*. Springer. 2017, pp. 97–117.

[39] Kyle D Julian et al. “Policy compression for aircraft collision avoidance systems”. In: *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*. IEEE. 2016, pp. 1–10.

[40] Matthew L. Daggitt et al. *Vehicle: Interfacing Neural Network Verifiers with Interactive Theorem Provers*. 2022. DOI: 10.48550/ARXIV.2202.05207. URL: https://arxiv.org/abs/2202.05207.

[41] Matthew L. Daggitt, Wen Kokke, Ekaterina Komendantskaya, et al. "The Vehicle Tutorial: Neural Network Verification with Vehicle". In: *Proceedings of the 6th Workshop on Formal Methods for ML-Enabled Autonomous Systems, FoMLAS@CAV*. Vol. 16. 2023.

[42] Matthew L Daggitt et al. "Vehicle: Bridging the Embedding Gap in the Verification of Neuro-Symbolic Programs". In: *arXiv preprint arXiv:2401.06379* (2024).

[43] Colin Kessler et al. "Neural Network Verification for Gliding Drone Control: A Case Study". In: *arXiv preprint arXiv:2505.00622* (2025).

[44] Xiaowei Huang et al. *Safety Verification of Deep Neural Networks*. 2017. arXiv: 1610.06940 [cs.AI].

[45] Guy Katz et al. "The Marabou Framework for Verification and Analysis of Deep Neural Networks". In: July 2019, pp. 443–452. ISBN: 978-3-030-25539-8.

[46] Eric Wong and Zico Kolter. "Provable defenses against adversarial examples via the convex outer adversarial polytope". In: *International conference on machine learning*. PMLR. 2018, pp. 5286–5295.

[47] Timon Gehr et al. "Ai2: Safety and robustness certification of neural networks with abstract interpretation". In: *2018 IEEE symposium on security and privacy (SP)*. IEEE. 2018, pp. 3–18.

[48] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. "Certified adversarial robustness via randomized smoothing". In: *international conference on machine learning*. PMLR. 2019, pp. 1310–1320.

[49] Arthur S Jacobs et al. "AI/ML for Network Security: The Emperor has no Clothes". In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2022, pp. 1537–1551.

[50] Marco Casadio et al. "Neural Network Robustness as a Verification Property: A Principled Case Study". In: *Computer Aided Verification (CAV 2022)*. Lecture Notes in Computer Science. Springer, 2022.

[51] Zico Kolter and Aleksander Madry. "Adversarial robustness: Theory and practice". In: *Tutorial at NeurIPS* (2018), p. 3.

[52] Sylvestre-Alvise Rebuffi et al. "Data augmentation can improve robustness". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 29935–29948.

[53] Natalia Slusarz et al. "Logic of Differentiable Logics: Towards a Uniform Semantics of DL". In: *LPAR 2023: Proceedings of 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Manizales, Colombia, 4-9th June 2023*. Ed. by Ruzica Piskac and Andrei Voronkov. Vol. 94. EPiC Series in Computing. EasyChair, 2023, pp. 473–493. DOI: 10.29007/C1NT. URL: https://doi.org/10.29007/c1nt.

[54] Sven Gowal et al. "Scalable verified training for provably robust image classification". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 4842–4851.

[55] Huan Zhang et al. "Towards stable and efficient training of verifiably robust neural networks". In: *8th International Conference on Learning Representations, ICLR 2020*. 2020.

[56] Mark Niklas Müller et al. *Certified Training: Small Boxes are All You Need*. 2023. arXiv: 2210.04871 [cs.LG].

[57] Yuhao Zhang, Aws Albarghouthi, and Loris D'Antoni. "Robustness to programmable string transformations via augmented abstract training". In: *Proceedings of the 37th International Conference on Machine Learning*. 2020, pp. 11023–11032.

[58] Daniel Kroening and Wolfgang Paul. "Automated Pipeline Design". In: *Proc. of 38th ACM/IEEE Design Automation Conference (DAC 2001)*. ACM Press, 2001, pp. 810–815.

[59] Vishnu A Patankar, Alok Jain, and Randal E Bryant. "Formal verification of an ARM processor". In: *Proceedings Twelfth International Conference on VLSI Design.(Cat. No. PR00013)*. IEEE. 1999, pp. 282–287.

[60] Jacques-Henri Jourdan et al. "A formally-verified C static analyzer". In: *ACM SIGPLAN Notices* 50.1 (2015), pp. 247–259.

[61] Roberto Metere and Luca Arnaboldi. “Automating cryptographic protocol language generation from structured specifications”. In: *Proceedings of the IEEE/ACM 10th International Conference on Formal Methods in Software Engineering*. 2022, pp. 91–101.

[62] Jim Woodcock et al. “Formal methods: Practice and experience”. In: *ACM computing surveys (CSUR)* 41.4 (2009), pp. 1–36.

[63] Stanley Bak, Changliu Liu, and Taylor Johnson. “The second international verification of neural networks competition (vnn-comp 2021): Summary and results”. In: *arXiv preprint arXiv:2109.00498* (2021).

[64] Teodora Baluta et al. “Scalable quantitative verification for deep neural networks”. In: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE. 2021, pp. 312–323.

[65] Changliu Liu et al. “Algorithms for verifying deep neural networks”. In: *Foundations and Trends® in Optimization* 4.3-4 (2021), pp. 244–404.

[66] Gagandeep Singh et al. “Beyond the single neuron convex barrier for neural network certification”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[67] Clark Barrett and Cesare Tinelli. “Satisfiability modulo theories”. In: *Handbook of model checking* (2018), pp. 305–343.

[68] Aws Albarghouthi et al. “Introduction to neural network verification”. In: *Foundations and Trends® in Programming Languages* 7.1–2 (2021), pp. 1–157.

[69] George Dantzig. *Linear programming and extensions*. Princeton university press, 1963.

[70] Wayne L Winston. *Operations research: applications and algorithm*. Thomson Learning, Inc., 2004.

[71] Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. “Maximum resilience of artificial neural networks”. In: *Automated Technology for Verification and Analysis: 15th International Symposium, ATVA 2017, Pune, India, October 3–6, 2017, Proceedings 15*. Springer. 2017, pp. 251–268.

[72] Alessio Lomuscio and Lalit Maganti. “An approach to reachability analysis for feed-forward relu neural networks”. In: *arXiv preprint arXiv:1706.07351* (2017).

[73] Vincent Tjeng, Kai Xiao, and Russ Tedrake. "Evaluating robustness of neural networks with mixed integer programming". In: *n International Conference on Learning Representations,* (2019).

[74] Gagandeep Singh et al. "Fast and Effective Robustness Certification". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper/2018/file/f2f446980d8e971ef3da97af089481c3-Paper.pdf.

[75] LLC Gurobi Optimization. "gurobi: Gurobi Optimizer 9.1 interface". In: *R package version* (2020), pp. 9–1.

[76] Patrick Cousot and Radhia Cousot. "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints". In: *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. 1977, pp. 238–252.

[77] Patrick Cousot. "Verification by abstract interpretation". In: *Verification: Theory and Practice: Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday*. Springer, 2003, pp. 243–268.

[78] Zhaoyang Lyu et al. "Fastened crown: Tightened neural network robustness certificates". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04. 2020, pp. 5037–5044.

[79] Matthew Mirman, Timon Gehr, and Martin Vechev. "Differentiable abstract interpretation for provably robust neural networks". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 3578–3586.

[80] Linyi Li, Tao Xie, and Bo Li. "Sok: Certified robustness for deep neural networks". In: *2023 IEEE symposium on security and privacy (SP)*. IEEE. 2023, pp. 1289–1310.

[81] Huan Zhang et al. "Efficient neural network robustness certification with general activation functions". In: *Advances in neural information processing systems* 31 (2018).

[82] Mark Niklas Müller et al. "PRIMA: general and precise neural network certification via scalable convex hull approximations." In: *Proc. ACM Program. Lang.* 6.POPL (2022), pp. 1–33.

[83] Rudy R Bunel et al. “A unified view of piecewise linear neural network verification”. In: *Advances in Neural Information Processing Systems* 31 (2018).

[84] Rudy Bunel et al. “Branch and bound for piecewise linear neural network verification”. In: *Journal of Machine Learning Research* 21.2020 (2020).

[85] Claudio Ferrari et al. “Complete Verification via Multi-Neuron Relaxation Guided Branch-and-Bound”. In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=l_amHf1oaK.

[86] Matt Jordan, Justin Lewis, and Alexandros G Dimakis. “Provable certificates for adversarial examples: Fitting a ball in the union of polytopes”. In: *Advances in neural information processing systems* 32 (2019).

[87] Huan Zhang et al. “General cutting planes for bound-propagation-based neural network verification”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 1656–1670.

[88] Mark Niklas Müller et al. *PRIMA: Precise and General Neural Network Certification via Multi-Neuron Convex Relaxations*. 2021. arXiv: 2103.03638 [cs.AI].

[89] Mathias Lecuyer et al. “Certified robustness to adversarial examples with differential privacy”. In: *2019 IEEE symposium on security and privacy (SP)*. IEEE. 2019, pp. 656–672.

[90] Bai Li et al. “Certified adversarial robustness with additive noise”. In: *Advances in neural information processing systems* 32 (2019).

[91] Krishnamurthy Dj Dvijotham et al. “A framework for robustness certification of smoothed classifiers using f-divergences”. In: (2020).

[92] Dinghuai Zhang et al. “Black-box certification with randomized smoothing: A functional optimization based framework”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 2316–2326.

[93] Hadi Salman et al. “Provably robust deep learning via adversarially trained smoothed classifiers”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[94] Jeet Mohapatra et al. “Higher-order certification for randomized smoothing”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 4501–4511.

[95] Nicolas Papernot et al. “Distillation as a defense to adversarial perturbations against deep neural networks”. In: *2016 IEEE symposium on security and privacy (SP)*. IEEE. 2016, pp. 582–597.

[96] Stephan Zheng et al. “Improving the robustness of deep neural networks via stability training”. In: *Proceedings of the ieee conference on computer vision and pattern recognition*. 2016, pp. 4480–4488.

[97] Nicholas Carlini and David Wagner. “Towards evaluating the robustness of neural networks”. In: *2017 ieee symposium on security and privacy (sp)*. Ieee. 2017, pp. 39–57.

[98] Anish Athalye, Nicholas Carlini, and David Wagner. “Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples”. In: *International conference on machine learning*. PMLR. 2018, pp. 274–283.

[99] Sven Gowal et al. *On the Effectiveness of Interval Bound Propagation for Training Verifiably Robust Models*. 2019. arXiv: 1810.12715 [cs.LG].

[100] Aditi Raghunathan et al. “Adversarial training can hurt generalization”. In: *arXiv preprint arXiv:1906.06032* (2019).

[101] Mislav Balunovic and Martin Vechev. “Adversarial Training and Provable Defenses: Bridging the Gap”. In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=SJxSDxrKDr.

[102] Wei Emma Zhang et al. “Adversarial attacks on deep-learning models in natural language processing: A survey”. In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 11.3 (2020), pp. 1–41.

[103] Wenqi Wang et al. “Towards a Robust Deep Neural Network against Adversarial Texts: A Survey”. In: *IEEE Transactions on Knowledge and Data Engineering* (2021), pp. 1–1.

[104] Xuezhi Wang, Haohan Wang, and Diyi Yang. “Measure and Improve Robustness in NLP Models: A Survey”. In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2022, pp. 4569–4586.

[105] Zongyi Li et al. “Searching for an Effective Defender: Benchmarking Defense against Adversarial Word Substitution”. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 2021, pp. 3137–3147.

[106] Yi Zhou et al. “Defense against synonym substitution-based adversarial attacks via Dirichlet neighborhood ensemble”. In: *Association for Computational Linguistics (ACL)*. 2021.

[107] Chen Zhu et al. “FreeLB: Enhanced Adversarial Training for Natural Language Understanding”. In: *International Conference on Learning Representations*. 2019.

[108] Xinshuai Dong et al. “Towards robustness against natural language word substitutions”. In: *arXiv preprint arXiv:2107.13541* (2021).

[109] Steven Y. Feng et al. “A Survey of Data Augmentation Approaches for NLP”. In: *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*. Online: Association for Computational Linguistics, 2021, pp. 968–988. DOI: 10.18653/v1/2021.findings-acl.84. URL: https://aclanthology.org/2021.findings-acl.84.

[110] Kaustubh D. Dhole et al. “NL-Augmenter: A Framework for Task-Sensitive Natural Language Augmentation”. In: *CoRR* abs/2112.02721 (2021). arXiv: 2112.02721. URL: https://arxiv.org/abs/2112.02721.

[111] Yong Cheng, Lu Jiang, and Wolfgang Macherey. “Robust Neural Machine Translation with Doubly Adversarial Inputs”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 2019, pp. 4324–4333.

[112] Mohit Iyyer et al. “Adversarial Example Generation with Syntactically Controlled Paraphrase Networks”. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. 2018, pp. 1875–1885.

[113] Yu Cao et al. “TASA: Deceiving Question Answering Models by Twin Answer Sentences Attack”. In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. 2022, pp. 11975–11992.

[114] Bin Liang et al. “Deep text classification can be fooled”. In: *arXiv preprint arXiv:1704.08006* (2017).

[115] Javid Ebrahimi et al. “HotFlip: White-Box Adversarial Examples for Text Classification”. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. 2018, pp. 31–36.

[116] Yibin Lei et al. “Phrase-level Textual Adversarial Attack with Label Preservation”. In: *Findings of the Association for Computational Linguistics: NAACL 2022*. 2022, pp. 1095–1112.

[117] Yonatan Belinkov and Yonatan Bisk. “Synthetic and natural noise both break neural machine translation”. In: *arXiv preprint arXiv:1711.02173* (2017).

[118] Ji Gao et al. “Black-box generation of adversarial text sequences to evade deep learning classifiers”. In: *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE. 2018, pp. 50–56.

[119] Jinfeng Li et al. “TextBugger: Generating Adversarial Text Against Real-world Applications”. In: *Proceedings 2019 Network and Distributed System Security Symposium* (2019).

[120] Suranjana Samanta and Sameep Mehta. *Towards Crafting Text Adversarial Samples*. 2017. arXiv: 1707.02812 [cs.LG].

[121] Adam Ivankay et al. “Fooling explanations in text classifiers”. In: *arXiv preprint arXiv:2206.03178* (2022).

[122] Di Jin et al. “Is bert really robust? a strong baseline for natural language attack on text classification and entailment”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 34. 05. 2020, pp. 8018–8025.

[123] Moustafa Alzantot et al. “Generating Natural Language Adversarial Examples”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 2018, pp. 2890–2896.

[124] Volodymyr Kuleshov et al. “Adversarial examples for natural language classification problems”. In: (2018).

[125] Jeffrey Pennington, Richard Socher, and Christopher D Manning. “Glove: Global vectors for word representation”. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[126] Milad Moradi and Matthias Samwald. “Evaluating the Robustness of Neural Language Models to Input Perturbations”. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 2021, pp. 1558–1570.

[127] Robin Jia and Percy Liang. “Adversarial Examples for Evaluating Reading Comprehension Systems”. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 2017, pp. 2021–2031.

[128] Yicheng Wang and Mohit Bansal. “Robust Machine Comprehension Models via Adversarial Training”. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. 2018, pp. 575–581.

[129] Tongshuang Wu et al. “Polyjuice: Generating Counterfactuals for Explaining, Evaluating, and Improving Models”. In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 2021, pp. 6707–6723.

[130] Wei-Lin Chiang et al. *Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality*. Mar. 2023. URL: https://lmsys.org/blog/2023-03-30-vicuna/.

[131] Boxin Wang et al. “Adversarial glue: A multi-task benchmark for robustness evaluation of language models”. In: *arXiv preprint arXiv:2111.02840* (2021).

[132] Robin Jia et al. “Certified Robustness to Adversarial Word Substitutions”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 2019, pp. 4129–4142.

[133] Po-Sen Huang et al. “Achieving Verified Robustness to Symbol Substitutions via Interval Bound Propagation”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 2019, pp. 4083–4093.

[134] Johannes Welbl et al. “Towards verified robustness under text deletion interventions”. In: (2020).

[135] Yuhao Zhang, Aws Albarghouthi, and Loris D’Antoni. “Certified Robustness to Programmable Transformations in LSTMs”. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 2021, pp. 1068–1083.

[136] Yibin Wang et al. “Robustness-Aware Word Embedding Improves Certified Robustness to Adversarial Word Substitutions”. In: *Findings of the Association for Computational Linguistics: ACL 2023*. 2023, pp. 673–687.

[137] Ching-Yun Ko et al. “POPQORN: Quantifying robustness of recurrent neural networks”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 3468–3477.

[138] Zhouxing Shi et al. *Robustness Verification for Transformers*. 2020. arXiv: 2002.06622 [cs.LG].

[139] Tianyu Du et al. “Cert-RNN: Towards Certifying the Robustness of Recurrent Neural Networks.” In: *CCS* 21.2021 (2021), pp. 15–19.

[140] Gregory Bonaert et al. “Fast and precise certification of transformers”. In: *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. 2021, pp. 466–481.

[141] Mao Ye, Chengyue Gong, and Qiang Liu. “SAFER: A Structure-free Approach for Certified Robustness to Adversarial Word Substitutions”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2020, pp. 3465–3475.

[142] Wenjie Wang et al. “Certified Robustness to Word Substitution Attack with Differential Privacy”. In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Ed. by Kristina Toutanova et al. Online: Association for Computational Linguistics, June 2021, pp. 1102–1112. DOI: 10.18653/v1/2021.naacl-main.87. URL: https://aclanthology.org/2021.naacl-main.87.

[143] Haiteng Zhao et al. “Certified robustness against natural language attacks by causal intervention”. In: *International Conference on Machine Learning*. PMLR. 2022, pp. 26958–26970.

[144] Jiehang Zeng et al. “Certified robustness to text adversarial attacks by randomized [mask]”. In: *Computational Linguistics* 49.2 (2023), pp. 395–427.

[145] Muchao Ye et al. “UniT: A Unified Look at Certified Robust Training against Text Adversarial Perturbation”. In: *Thirty-seventh Conference on Neural Information Processing Systems*. 2023.

[146] Xinyu Zhang et al. “Text-CRS: A Generalized Certified Robustness Framework against Textual Adversarial Attacks”. In: *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society. 2023, pp. 53–53.

[147] Zhen Zhang et al. “Certified Robustness for Large Language Models with Self-Denoising”. In: *arXiv preprint arXiv:2307.07171* (2023).

[148] Khalil Ghorbal, Eric Goubault, and Sylvie Putot. “The zonotope abstract domain taylor1+”. In: *Computer Aided Verification: 21st International Conference, CAV 2009, Grenoble, France, June 26-July 2, 2009. Proceedings 21*. Springer. 2009, pp. 627–633.

[149] Samuel R. Bowman et al. “A large annotated corpus for learning natural language inference”. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Ed. by Lluís Màrquez, Chris Callison-Burch, and Jian Su. Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 632–642. DOI: 10.18653/v1/D15-1075. URL: https://aclanthology.org/D15-1075.

[150] cjadams Jeffrey Sorensen Julia Elliott Lucas Dixon Mark McDonald nithum and Will Cukierski. *Toxic Comment Classification Challenge*. 2017. URL: https://kaggle.com/competitions/jigsaw-toxic-comment-classification-challenge.

[151] Andrew L. Maas et al. “Learning Word Vectors for Sentiment Analysis”. In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Ed. by Dekang Lin, Yuji Matsumoto, and Rada Mihalcea. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 142–150. URL: https://aclanthology.org/P11-1015.

[152] Richard Socher et al. “Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank”. In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Ed. by David Yarowsky et al. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1631–1642. URL: https://aclanthology.org/D13-1170.

[153] Tianxiao Shen et al. “Style transfer from non-parallel text by cross-alignment”. In: *Advances in neural information processing systems* 30 (2017).

[154] Bo Pang and Lillian Lee. "Seeing Stars: Exploiting Class Relationships for Sentiment Categorization with Respect to Rating Scales". In: *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*. Ed. by Kevin Knight, Hwee Tou Ng, and Kemal Oflazer. Ann Arbor, Michigan: Association for Computational Linguistics, June 2005, pp. 115–124. DOI: 10.3115/1219840.1219855. URL: https://aclanthology.org/P05-1015.

[155] Julian McAuley and Jure Leskovec. "Hidden factors and hidden topics: understanding rating dimensions with review text". In: *Proceedings of the 7th ACM conference on Recommender systems*. 2013, pp. 165–172.

[156] Adina Williams, Nikita Nangia, and Samuel Bowman. "A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Ed. by Marilyn Walker, Heng Ji, and Amanda Stent. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 1112–1122. DOI: 10.18653/v1/N18-1101. URL: https://aclanthology.org/N18-1101.

[157] Xiang Zhang, Junbo Zhao, and Yann LeCun. "Character-level convolutional networks for text classification". In: *Advances in neural information processing systems* 28 (2015).

[158] Xin Li and Dan Roth. "Learning Question Classifiers". In: *COLING 2002: The 19th International Conference on Computational Linguistics*. 2002. URL: https://aclanthology.org/C02-1150.

[159] Christina Montgomery. *Hearing on "Oversight of AI: Rules for Artificial Intelligence"*. https://www.ibm.com/policy/wp-content/uploads/2023/05/Christina-Montgomery-Senate-Judiciary-Testimony-5-16-23.pdf. Accessed: 2023-06-01. 2023.

[160] California State Legislature. "California senate bill no. 1001". In: 2018. URL: https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=201720180SB1001.

[161] Mauritz Kop. *EU Artificial Intelligence Act: The European Approach to AI*. 2021. URL: https://futurium.ec.europa.eu/sites/default/files/2021-

10/Kop%5C_EU%20Artificial%20Intelligence%20Act%20-%20The%20European%20Approach%20to%20AI%5C_21092021%5C_0.pdf.

[162] Timothy W Bickmore et al. "Patient and consumer safety risks when using conversational assistants for medical information: an observational study of Siri, Alexa, and Google Assistant". In: *Journal of medical Internet research* 20.9 (2018), e11510.

[163] Emily Dinan et al. "SafetyKit: First Aid for Measuring Safety in Open-domain Conversational Systems". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 4113–4133. DOI: 10.18653/v1/2022.acl-long.284. URL: https://aclanthology.org/2022.acl-long.284.

[164] Robin Sommer and Vern Paxson. "Outside the closed world: On using machine learning for network intrusion detection". In: *2010 IEEE symposium on security and privacy*. IEEE. 2010, pp. 305–316.

[165] MohammadNoor Injadat et al. "Multi-stage optimized machine learning framework for network intrusion detection". In: *IEEE Transactions on Network and Service Management* 18.2 (2020), pp. 1803–1816.

[166] Elie Alhajjar, Paul Maxwell, and Nathaniel Bastian. "Adversarial machine learning in network intrusion detection systems". In: *Expert Systems with Applications* 186 (2021), p. 115782.

[167] Roberto Doriguzzi-Corin et al. "LUCID: A practical, lightweight deep learning solution for DDoS attack detection". In: *IEEE Transactions on Network and Service Management* 17.2 (2020), pp. 876–889.

[168] Wei Wang et al. "HAST-IDS: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection". In: *IEEE access* 6 (2017), pp. 1792–1806.

[169] Zhiyuan Tan et al. "Detection of denial-of-service attacks based on computer vision techniques". In: *IEEE transactions on computers* 64.9 (2014), pp. 2519–2533.

[170] Yisroel Mirsky et al. "Kitsune: an ensemble of autoencoders for online network intrusion detection". In: *arXiv preprint arXiv:1802.09089* (2018).

[171] Andrea Venturi et al. “ARGANIDS: a novel network intrusion detection system based on adversarially regularized graph autoencoder”. In: *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*. 2023, pp. 1540–1548.

[172] Wai Weng Lo et al. “E-graphsage: A graph neural network based intrusion detection system for iot”. In: *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. IEEE. 2022, pp. 1–9.

[173] Chaoyun Zhang, Xavier Costa-Perez, and Paul Patras. “Adversarial attacks against deep learning-based network intrusion detection systems and defense mechanisms”. In: *IEEE/ACM Transactions on Networking* 30.3 (2022), pp. 1294–1311.

[174] Jianbo Chen, Michael I Jordan, and Martin J Wainwright. “Hopskipjumpattack: A query-efficient decision-based attack”. In: *2020 IEEE Symposium on Security and Srivacy (SP)*. IEEE. 2020, pp. 1277–1294.

[175] Florian Tramèr et al. “Ensemble adversarial training: Attacks and defenses”. In: *arXiv preprint arXiv:1705.07204* (2017).

[176] Lisa Liu et al. “Error Prevalence in NIDS Datasets: A Case Study on CIC-IDS-2017 and CSE-CIC-IDS-2018”. In: *2022 IEEE Conference on Communications and Network Security (CNS)*. IEEE. 2022, pp. 254–262.

[177] Robert Flood, Gints Engelen, David Aspinall, et al. “Bad Design Smells in Benchmark NIDS Datasets”. In: *2024 IEEE 9th European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2024, pp. 658–675.

[178] Marta Catillo et al. “A Critique on the Use of Machine Learning on Public Datasets for Intrusion Detection”. In: *International Conference on the Quality of Information and Communications Technology*. Springer. 2021, pp. 253–266.

[179] Giovanni Apruzzese, Luca Pajola, and Mauro Conti. “The cross-evaluation of machine learning-based network intrusion detection systems”. In: *IEEE Transactions on Network and Service Management* 19.4 (2022), pp. 5152–5169.

[180] Kai Wang et al. “BARS: Local Robustness Certification for Deep Learning based Traffic Analysis Systems.” In: *NDSS*. 2023.

[181] Yaniv Leviathan and Yossi Matias. “Google Duplex: An AI System for Accomplishing Real World Tasks Over the Phone”. In: *Google AI Blog* (2018).

[182] Johnny Lieu. *Google's creepy AI phone call feature will disclose it's a robot, after backlash*. https://mashable.com/2018/05/11/google-duplex-disclosures-robot. Mashable. Accessed 2023-03-16. 2018.

[183] Michael Atleson. *Chatbots, deepfakes, and voice clones: AI deception for sale*. https://www.ftc.gov/business-guidance/blog/2023/03/chatbots-deepfakes-voice-clones-ai-deception-sale. Federal Trade Commission. Accessed: 2023-06-16. 2023.

[184] World Economic Forum. *Chatbots RESET: A Framework for Governing Responsible Use of Conversational AI in Healthcare*. https://www.weforum.org/publications/chatbots-reset-a-framework-for-governingresponsible-use-of-conversational-ai-in-healthcare/. Accessed 2023-06-19. 2020.

[185] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL].

[186] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. "The Quickhull algorithm for convex hulls". In: *ACM TRANSACTIONS ON MATHEMATICAL SOFTWARE* 22.4 (1996), pp. 469–483.

[187] Hossein Sartipizadeh and Tyrone L. Vincent. *Computing the Approximate Convex Hull in High Dimensions*. 2016. arXiv: 1603.04422 [cs.CG].

[188] Yuting Jia et al. *On Geometric Structure of Activation Spaces in Neural Networks*. 2019. arXiv: 1904.01399 [cs.LG].

[189] Virginia Klema and Alan Laub. "The singular value decomposition: Its computation and some applications". In: *IEEE Transactions on automatic control* 25.2 (1980), pp. 164–176.

[190] Gagandeep Singh et al. *Replication Package for the Article: An Abstract Domain for Certifying Neural Networks*.

[191] Marco Casadio et al. "ANTONIO: Towards a Systematic Method of Generating NLP Benchmarks for Verification". In: *Proceedings of the 6th Workshop on Formal Methods for ML-Enabled Autonomous Systems*. Ed. by Nina Narodytska et al. Vol. 16. Kalpa Publications in Computing. EasyChair, 2023, pp. 59–70. DOI: 10.29007/7wxb. URL: https://easychair.org/publications/paper/9ZGS.

[192] Edward Beeching et al. *Open LLM Leaderboard*. https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard. 2023.

[193] Stephanie Lin, Jacob Hilton, and Owain Evans. “TruthfulQA: Measuring How Models Mimic Human Falsehoods”. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2022, pp. 3214–3252.

[194] Wenhui Wang et al. “MiniLMv2: Multi-Head Self-Attention Relation Distillation for Compressing Pretrained Transformers”. In: *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*. 2021, pp. 2140–2151.

[195] Lu Yu and Verena Rieser. *Adversarial Robustness of Visual Dialog*. 2022. arXiv: 2207.02639 [cs.CV].

[196] David Liljequist, Britt Elfving, and Kirsti Skavberg Roaldsen. “Intraclass correlation–A discussion and demonstration of basic features”. In: *PloS one* 14.7 (2019), e0219854.

[197] Chin-Yew Lin. “ROUGE: A Package for Automatic Evaluation of Summaries”. In: *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 74–81. URL: https://aclanthology.org/W04-1013.

[198] Yuli Vasiliev. *Natural language processing with Python and spaCy: A practical introduction*. No Starch Press, 2020.

[199] Lucas C Cordeiro et al. *Neural Network Verification is a Programming Language Challenge*.

[200] Christopher Brix et al. “The fourth international verification of neural networks competition (vnn-comp 2023): Summary and results”. In: *arXiv preprint arXiv:2312.16760* (2023).

[201] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “" Why should i trust you?" Explaining the predictions of any classifier”. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1135–1144.

[202] Scott M Lundberg and Su-In Lee. “A unified approach to interpreting model predictions”. In: *Advances in neural information processing systems* 30 (2017).

[203] Henry Clausen, Robert Flood, and David Aspinall. “Traffic generation using containerization for machine learning”. In: *Proceedings of the 2019 Workshop on DYnamic and Novel Advances in Machine Learning and Intelligent Cyber Security*. 2019, pp. 1–12.