Verifying Explanations for Neural Networks

by

Nham Le

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2024

© Nham Le 2024

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Taylor T. Johnson Associate Professor Department of Computer Science Vanderbilt University
Supervisor(s):	Arie Gurfinkel Professor Department of Electrical and Computer Engineering University of Waterloo
Internal Member:	Derek Rayside Associate Professor Department of Electrical and Computer Engineering University of Waterloo
Internal-External Member:	Richard Trefler Associate Professor David R. Cheriton School of Computer Science University of Waterloo
Other Member(s):	Yash Vardhan Pant Assistant Professor Department of Electrical and Computer Engineering University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

Contents and results presented in this thesis are collaborations with my supervisor, Professor Arie Gurfinkel, together with our collaborators.

- Chapter 3 was done with Professor Xujie Si from University of Toronto and his students: Chuqin Geng, Xiaojie Xu, and Zhaoyue Wan.
- Chapter 4 was done during my two internships at Amazon Automated Reasoning Group where I lead the designing of the system and the development and testing of different clause sharing strategies. Pei-Wei Chen and Kunal Sheth worked on the gRPC and hardening of the system. Clark Barrett, Bruno Dutertre, Robert Jones, Andrew Reynolds, Chriss Stevens, Byron Cook, and Mike Whalen were managers and technical advisors of the project.
- Chapter 5 was done with Professor Xujie Si from University of Toronto, and his student Chuqin Geng.
- Chapter 6 was done with our lab's Undergraduate Research Assistant Henry Guo, Professor Xujie Si from University of Toronto, and his student Chuqin Geng.

Abstract

Deep neural networks (DNNs) have been applied in solving many complex tasks across multiple domains, many of which have direct effects on our daily lives: generative models are replacing traditional search engines for answering questions, cars are being driven by neural networks, doctors and radiologists are using neural nets to diagnose patients more efficiently, financial systems are run by automated trading bots, etc. Coupled with the ever-increasing of DNNs' complexity, the need for explaining their predictions and verifying their safety is clear.

Generally speaking, verifying a DNN involves checking if it behaves as expected for unseen inputs in a particular region, and explaining a DNN involves interpreting the network's prediction on a given input. Both approaches have their own pros and cons: the output of any input in a verified region is proven to be correct (with respect to a specification), but such regions are minuscule compared to the whole input space, not just because of the performance of the tools, but because of the inherent limits in ϵ-robustness – the commonly used verification specifications; and while explanation methods can be applied to explain the output given any input, they are post-hoc and hard to judge: does an explanation make sense because the DNN is working close to how a human being process the same input, or because the explanation visualizes the input itself without taking the model in consideration?

Our main insight: we can combine both verification and explanation, resulting in novel verification problems towards a robust explanation for neural networks. However, any verification problem (or specification) can not exist in isolation, but in a symbiosis relationship with the tools solving it. When we propose a new specification, it is expected that existing tools cannot solve it effectively, or may not work at all. Interesting problems push developers to improve the tools, and better tools widen the design space for researchers to come up with even more interesting specifications. Thus, in this proposal, we are introducing not just novel specifications, but how to solve them by building better tools.

This thesis presents a series of results and research ideas based on that insight. First, we show that by extending ϵ-robustness with an explanation function (the activation pattern of the DNN), we can verify a bigger region of the input space using existing verification tools. Second, by verifying the explanation functions, we provide a robust way to compare different explanation methods. Finally, even when the combination of existing DNNs' verification specifications and explanation functions is friendlier to existing verification tools, we still run into scalability issues as we increase the size of the networks. Thus, in this thesis we also present our results on building a distributed SMT solver, which lies at the heart of many neural network verification tools.

Acknowledgements

I would like to thank all the people who made this thesis possible.

Dedication

First and foremost, I would like to express my deepest gratitude to my advisor, Arie Gurfinkel, whose unwavering support and guidance have been the cornerstone of this journey. Without his encouragement and mentorship, this thesis would not have been possible.

I am also deeply thankful to my thesis committee members: Derek Rayside, Richard Trefler, Yash Vardhan Pant, and Taylor T. Johnson. Your meticulous reading and insightful suggestions have significantly improved this work. To Derek, working with you on the Introduction to Compiler course has been an immense pleasure. You are not only a valued committee member but also a dear friend and neighbor in Toronto.

I extend my heartfelt thanks to all my collaborators, especially Xujie Si, Robert Jones, and Bruno Dutertre, for the many productive discussions that have greatly contributed to this thesis. A special mention to Xujie Si: fortune has put us in the same room at Waterloo during the challenging times of COVID. You have been an exemplar of what a successful PhD journey looks like, and your achievements have inspired me through the most difficult phases of my own program. I am also grateful to Allen Geng, Clark Barrett, Henry Guo, Xiaojie Xu, Zhaoyue Wang, Byron Cook, Andrew Reynolds, Kunal Sheth, Pei-Wei Chen, and Chriss Stevens. This thesis is a testament to our collaborative efforts, and I am thankful for your contributions.

I dedicate this to Hung-Quan Tran, and Manh-Tuan Lai. You guys have shown me what I could be, rather than what I should be.

To the exceptional group of PhDs, Postdocs, and Masters students brought together by Arie: Hari and Jakub, you are more than colleagues; you are lifelong friends. Siddharth, Yusen, Isa, it has been an honor to know you. I am also grateful to my Vietnamese friends at the University of Waterloo—Thi-Xuan Vu, Trang Bui, Hoang-Linh Nguyen, and Huy Hoang—for the wonderful memories we have shared.

I dedicate a special part of this acknowledgment to the incredible team at Amazon Minneapolis. While I may have only been with you for three months, you became the work family I never had. My heartfelt thanks to Mike Whalen, Andrew Gacek, Dan DaCosta, John Backes, Mary Southern, and Amanda Anderson. Amanda, your twenty-bucks note has been framed, and will forever be cherished. Minneapolis will always hold a special place in my heart.

I am truly blessed to have had my wife, Anh-Duong Nguyen, by my side throughout the highs and lows of this PhD journey. I am forever indebted to her for her unwavering support, companionship, and wise counsel during the darkest moments.

Finally, I would like to thank my parents and brother for their unconditional love and support at every stage of my life. No words can fully convey my gratitude to them. This thesis is dedicated to them.

And to Edwin Le, may you grow up, loved and happy. Uncle loves you.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, Deep Neural Networks (DNNs) have been applied successfully to a wide range of fields, including but not limited to computer vision [57, 130], natural language [18], robotics [112, 65], finance [59, 17], and health care [38, 131]. These applications have the potential to directly affect our lives, from supporting doctors in medical analysis to making millions of trading orders per second to responding to our voices, to driving us on highways. Thanks to their ability to learn complex patterns from a vast amount of data and generalize to unseen data, deep neural networks are the best-performing computing approach in many benchmarks, surpassing even human beings in tasks like image recognition [30, 66] or playing games [116].

However, despite DNNs' state-of-the-art performance, they occasionally make devastating mistakes: some caused by *bias* in the dataset (e.g gender or races), some caused by the inherent *brittleness* in the model. As DNNs are being used more and more in many high-stake real-world applications that directly impact various aspects of our lives, the challenge of validating the prediction of DNNs[68, 7] has attracted much attention from the research community.

The research area of validating the operation of DNNs, or *Safe and Explainable AI*, can be generalized in two camps: *explanation* (XAI) and *verification*. The formal aims to extract an easy-to-understand visualization of the operation of a neural network and then involve human judgment into the loop to validate why a prediction should be trusted, while the latter tries to use mathematics to automatically prove the correctness of a prediction given by a neural network, with respect to some formulations of "correctness".

Both approaches have their pros and cons. An explanation function (called *X-func* for short) is intuitive and user-friendly, and can be used to help human being to choose a

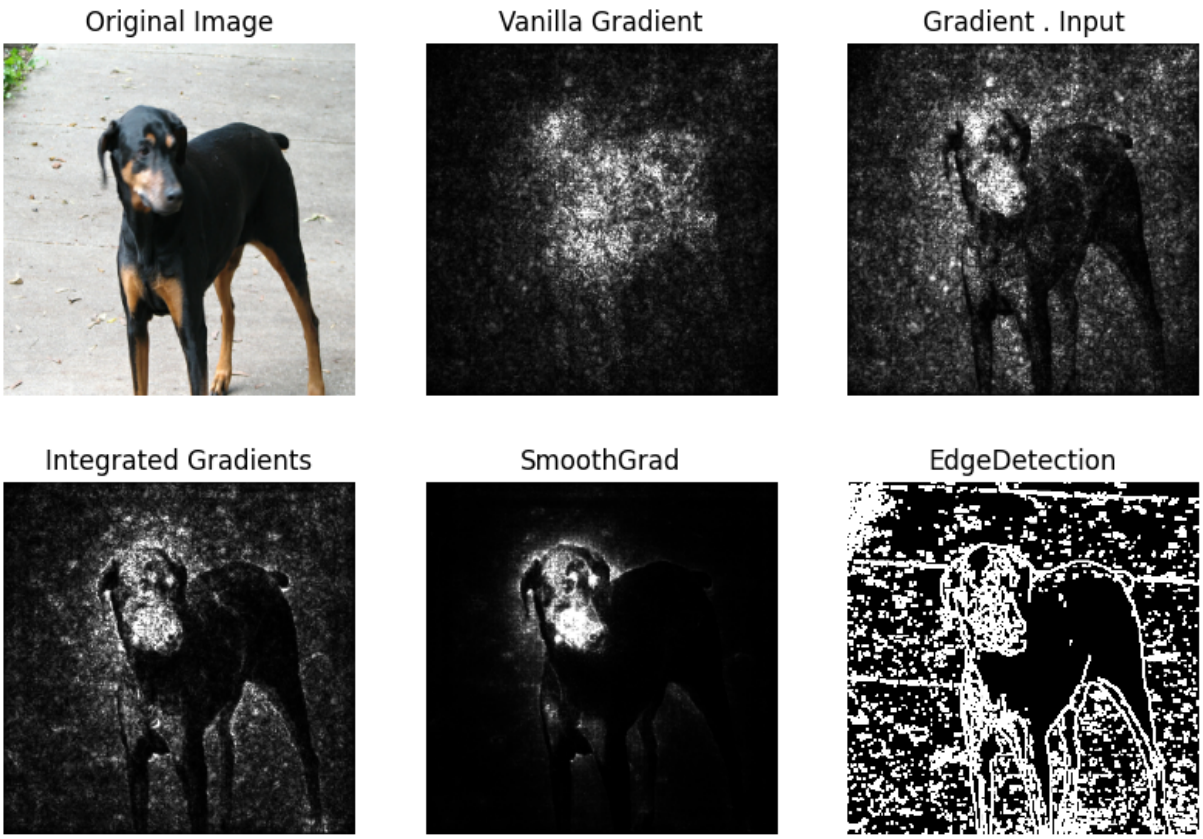
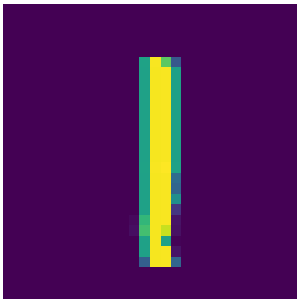


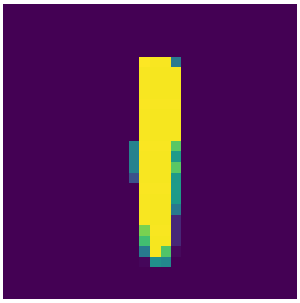
Figure 1.1: Visualizing the explanation given by some common methods, compared with visualizing a model-agnostic edge detector.

better model, detect irregularities, or debug the model [101]. However, to the best of our knowledge, none of the existing explanation techniques attempt to justify their *verifiablity* – if a single input x is classified by a neural net N as y because of a predicate reason $P(x)$ (written $P(x) \implies N(x) = y$), does that mean all inputs that satisfy P will have the same classification y? As a consequence, explanation methods may fall into the trap of explaining the input itself, instead of the network's operation given the input [2]. As an extreme example, Fig. 1.1 shows that an off-the-shelf edge detector that does no classification at all, looks shockingly convincing as a smart neural network!

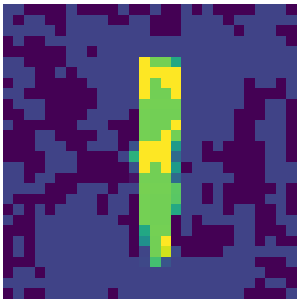
On the other hand, verification methods achieve verifiability by construction, but suffer from the difficulty in defining what is "correct". Most works [69, 70, 62, 60, 133] use the specification of *adversarial robustness* as a proxy: a NN should correctly classifies an



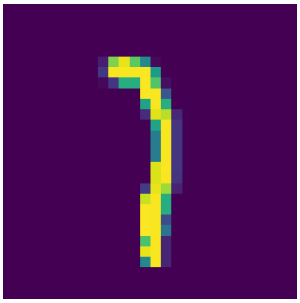
(a) A testing image from MNIST, classified as 1



(b) The closest training image in MNIST, whose L_∞ distance to Fig. 1.2a is 0.5294



(c) An adversarial example misclassified as 8, whose L_∞ distance to Fig. 1.2b is 0.2



(d) A testing image, on which our verified NAP (for digit 1) disagrees with the ground truth (i.e., 7)

Figure 1.2: The limitation of "data-as-specification": First three images show that a test input can be much further away (in L_∞) from its closest train input compared to adversarial examples (the upper bound of a verifiable local region). The last image shows that even data itself can be imperfect.

image as a given adversarial label under perturbations with a specific norm (usually l_∞). Generally speaking, existing works use a paradigm of *data as specification* — the robustness of local neighborhoods of reference data points with ground-truth labels is the only specification of correct behaviors. However, from a learning perspective, this would lead to *overfitted* specification, since only local neighborhoods of reference inputs get certified. As a concrete example, Figure 1.2 illustrates the fundamental limitation of such overfitted specifications. Specifically, a testing input like the one shown in Fig. 1.2a can never be verified even if all local neighborhoods of all training images have been certified using the L_∞ norm. This is because adversarial examples like Fig. 1.2c fall into a much closer region compared to testing inputs (e.g., Fig. 1.2a), as a result, the truly verifiable region for a given reference input like Fig. 1.2b can only be smaller.

Given the pros and cons of both approaches, one natural question arises: is this possible to combine the two to get the best of both worlds? At the heart of this thesis is the research question: How do we formulate and efficiently solve the problem of verifying the explanations for a neural network?

Challenges To instantiate the verification of explanation methods for neural networks, we need to answer three key questions. First, what are the interesting safety proper-

ties/specifications for *X-func*s? For example, given that a saliency map is often judged visually, how do we encode the notion of "visually good" for a saliency map? Ideally, we want the specifications to be meaningful, i.e close to the human interpretation of the *X-func*s, but also solvable (at least in theory) given the current state-of-the-art verification techniques. This thesis explores multiple such specifications for different explanation functions, namely the Neural Activation Functions (NAPs), the Vanilla Gradient (VG) saliency map, and Mechanical Interpretation (MI) for Large Language Models.

Second, given the specifications, how can we solve it using the existing tools? Most verification tools operate in two stages: encoding a problem into an intermediate representation and then solving the problem in that intermediate form. Consequently, verifying *X-func*s requires first addressing the encoding problem, which is non-trivial because all existing neural network verifiers utilize intermediate representations designed for encoding mainly the forward pass of the network. Once the encoding phase is complete, the challenges of solving the problem emerge: since the verification problem itself is generally undecidable, the effectiveness of existing tools in practice relies on domain-specific techniques developed over time. These heuristics may not be applicable to our problem, necessitating the development of novel techniques tailored to our specific challenge.

Third, how do we construct a more scalable solver when scalability challenges inevitably arise? State-of-the-art neural network verifiers possess inherent scalability limitations, and consequently, employing them in their current form to verify novel specifications is likely to be insufficient. Central to many, if not all, neural network verifiers is the Satisfiability Modulo Theory (SMT) solver. This thesis introduces our state-of-the-art distributed SMT solver, which establishes the groundwork for developing more efficient distributed neural verifiers.

Although seemingly unrelated at first glance, a common thread connecting the three challenges is the significant difficulty in scaling existing tools to effectively verify a substantial portion of the input space for a large deep neural network (DNN), considering commonly utilized specifications and the concept of "safety" (adversarial robustness). This thesis hypothesizes that by extending the existing specifications with an *explanation* function, verifying these new specifications becomes both more meaningful and more compatible with existing verification tools. However, any novel verification problem also necessitates the development of improved tools: the problem and its solvers interact symbiotically, with compelling problems driving developers to enhance tool efficiency, and better tools expanding the design space for researchers to conceive even more interesting and challenging specifications.

Contributions and Organizations In this thesis, we present a series of results that build up on both fronts – the specification and the tool.

- In Chapter 2, we briefly introduce necessary background on Neural Networks, Neural Network Explanation and Verification, and SMT solving.
- In Chapter 3, we study the MNIST and CIFAR10 dataset and argue that the widely used point-wise ϵ-robustness specification is limited: most real datapoints lie outside of the maximum verifiable region. Consequently, given that specification and any verification tool, even if we can verify the maximum area surrounding all datapoints in the dataset, any unseen datapoint will still be in an unverified region! In other words, no matter how good the tool is, we cannot verify any unseen datapoint. Thus, we propose a novel specification in which the specification looks at not only the datapoint, but also the activation pattern of the neural network for that datapoint.
- In Chapter 4, we build a distributed SMT solver based on cvc5- a state-of-the-art SMT solver. As we discuss in Chapter 2, many neural network verification problems can be viewed as SMT problems in Quantifier-free Linear Real Arithmetic (QF-LRA). Up until now, most SMT solvers are single-threaded, and thus cannot be scaled horizontally to solve a problem faster. Building an efficient distributed SMT solver is a major step toward building efficient neural network verifiers for bigger neural networks over larger input spaces.
- In Chapter 5, we study *saliency map* methods for DNNs and propose novel safety specifications for them. We focus on *Vanilla Gradient* – the earliest yet surprisingly effective saliency map. We argue that while our proposed specifications can be encoded and solved as a QF-LRA problem, off-the-shelf SMT solvers cannot solve them effectively. Thus, we propose a novel method combining both Abstraction-based and Constraint-based neural network verifiers to solve our specifications.
- In Chapter 6, we propose and verify novel safety properties for *mechanical interpretation* – the state-of-the-art explanation methods for large language models (LLMs). We contend that although the scale of LLMs often exceeds the verification capabilities of existing neural network verifiers, the apparatuses utilized in mechanical interpretation (probes) are of manageable scales and constitute intriguing targets for verification.
- We further outline a number of future extensions in Chapter 7, and conclude in Chapter 8.

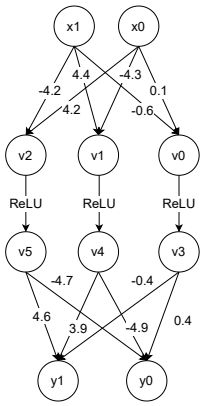
Chapter 2

Preliminaries

Verifying and Explaining Neural Networks is a rich research area that lies in the intersection of Deep Learning, Constraint solving, and Abstract Interpretation. In this chapter, we briefly introduce the most relevant background from each field to the thesis, as well as survey important related work. We also introduce the datasets and neural net architectures that we use throughout the thesis.

2.1 Targeted neural network architectures

Over the years, many neural network architectures have been introduced to solve different tasks: Feed-forward Neural Networks (FNNs) (including Convolutional Neural Networks [97]) process data in one direction, making them suitable for tasks like image recognition. Recurrent Neural Networks [113] (RNNs) handle sequential data, like text, with loops that allow memory of previous inputs; LSTMs [58] enhance this by retaining information over longer sequences. Generative Adversarial Networks [45] (GANs) create realistic data samples through a generator and discriminator. Graph Neural Networks [140] (GNNs) process and analyze data structured as graphs, capturing relationships and dependencies between nodes in tasks like node classification, link prediction, and graph classification. Transformer networks [130] excel in natural language processing by managing long-range dependencies and parallel computations, leading to the creation of Large Language Models (LLMs). In this thesis, we focus on FNNs and Transformer/LLMs.



(a) XNET: A NN that computes the analog XOR function.

$v_0 = 0.1x_0 - 0.6x_1$
$v_1 = -4.3x_0 + 4.4x_1$
$v_2 = 4.2x_0 - 4.2x_1$
$v_3 = max(v_0, 0)$
$v_4 = max(v_1, 0)$
$v_5 = max(v_2, 0)$
$y_0 = 0.4v_3 - 4.9v_4 + 3.9v_5 + 6.7$
$y_1 = -0.4v_3 + 3.9v_4 + 4.6v_5 - 7.4$
$x_0 \leq 0.1 \wedge x_0 \geq 0.02$
$x_1 \leq 0.1 \wedge x_1 \geq 0.02$
$0 < y_0 - y_1$

(b) Marabou's system of constraints for verifying that XNET is 0.04-robust at (0.06, 0.06)

$v_0 = 0.1x_0 - 0.6x_1$
$v_1 = -4.3x_0 + 4.4x_1$
$v_2 = 4.2x_0 - 4.2x_1$
$v_3 = v_0$
$v_4 = max(v_1, 0)$
$v_5 = 0$
$y_0 = 0.4v_4 - 4.9v_5 + 3.9v_6 + 6.7$
$y_1 = -0.4v_4 + 3.9v_5 + 4.6v_6 - 7.4$
$x_0 \leq 0.3 \wedge x_0 \geq 0$
$x_1 \leq 0.3 \wedge x_1 \geq 0$
$v_0 \geq 0$
$v_2 \leq 0$

(c) Check if $\mathcal{P}^1 = ((z_0), ())$ and $\mathcal{P}^0 = ((), (z_2))$ are non-ambiguous in the first quadrant using Marabou

Figure 2.1: Using Marabou to verify NAP properties of XNET.

2.1.1 Feed-forward Neural Networks

A feed-forward neural network $\mathcal{N}$ of L layers is a set $\{(\boldsymbol{W}^i, \boldsymbol{b}^i) \mid 1 \leq i \leq L\}$, where $\boldsymbol{W}^i$ and $\boldsymbol{b}^i$ are the weight matrix and the bias for layer i, respectively. The neural network $\mathcal{N}$ defined a function $F_N : \mathbb{R}^{d_0} \to \mathbb{R}^{d_L}$ (d_0 and d_L represent the input and output dimension, respectively), defined as $F_{\mathcal{N}}(x) = z^L(x)$, where$z^0(x) = x$, $z^i(x) = \boldsymbol{W}^i\sigma(z^{i-1}(x)) + \boldsymbol{b}^i$ and σ is the activation function. Neurons are indexed linearly by $v_0, v_1, \cdots$. In this thesis, we focus only on the ReLU activation function, i.e., $\sigma(x) = \max(x, 0)$ element-wise, but the idea and techniques can be generalized for different activation functions and architectures as well. The i^{th} element of the prediction vector $F_{\mathcal{N}}(x)[i]$ represents the score or likelihood for the i^{th} label, and the one with the highest score ($\arg\max_i F_{\mathcal{N}}(x)[i]$) is often considered as the predicted label of the network $\mathcal{N}$. We denote this output label as $\mathcal{O}_{\mathcal{N}}(x)$. When the context is clear, we omit the subscript $\mathcal{N}$ for simplicity.

A running example To help with illustrating later ideas, we present a two-layer feed-forward neural network XNET (Figure 2.1a) to approximate an analog XOR function $f(x_0, x_1) : [[0, 0.3] \cup [0.7, 1]]^2 \to \{0, 1\}$ such that $f(x_0, x_1) = 1$ iff $(x_0 \leq 0.3 \wedge x_1 \geq 0.7)$ or $(x_0 \geq 0.7 \wedge x_1 \leq 0.3)$. The network computes the function

$$F_{\text{XNET}}(x) = \boldsymbol{W}^1 \max(\boldsymbol{W}^0(x) + \boldsymbol{b}^0, 0) + \boldsymbol{b}^1$$

where $x = [x_0, x_1]$, and values of $\boldsymbol{W}^0, \boldsymbol{W}^1, \boldsymbol{b}^0, \boldsymbol{b}^1$ are shown in edges of Figure 2.1a. $\mathcal{O}(x) =$

0 if $F_{\text{XNET}}(x)[0] > F_{\text{XNET}}(x)[1]$, $\mathcal{O}(x) = 1$ otherwise.

Note that the weights and biases are not arbitrary. We have obtained it by constructing two sets of 1 000 randomly generated inputs, and training on one and validating on the other until the NN achieved a perfect F1-score of 1.

2.1.2 Large language models/The Transformer Architecture

In recent years, Large Language Models (LLMs) have been at the forefront of natural language processing (NLP) and artificial intelligence (AI). ChatGPT, arguably the most renowned LLM, has permeated the public consciousness, consistently making headlines due to its exceptional performance across a wide array of language tasks. At the heart of ChatGPT and all other LLMs is a neural network architecture pioneered by [130] – The Transformer. Although the architecture outlined in the original paper is designed for machine translation, adhering to an Encoder-Decoder framework, most subsequent large language models adopt a Decoder-only architecture, which we describe below.

The Decoder-only Transformer Architecture The Decoder-only Transformer architecture is a streamlined version of the original Transformer, focusing solely on the generative aspect of language modeling. It eliminates the need for an encoder, making it particularly suitable for tasks such as text generation, completion, and autoregressive modeling. Below is a formal description of its components and mathematical definitions.

- **Self-Attention Mechanism** The self-attention mechanism in a decoder-only Transformer computes a weighted sum of values, where the weights are determined by the similarity between queries and keys. Given an input sequence $X \in \mathbb{R}^{n \times d}$, where n is the sequence length and d is the dimension of the embeddings, the query Q, key K, and value V matrices are defined as:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

  where $W_Q, W_K, W_V \in \mathbb{R}^{d \times d_k}$ are learned projection matrices, and d_k is the dimension of the queries and keys.

  The attention scores are computed as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

To ensure causality (i.e., each position can only attend to earlier positions), a mask is applied:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + M\right)V$$

where M is a mask matrix with $M_{ij} = -\infty$ for $j > i$ and 0 otherwise, ensuring no information flow from future tokens.

- **Multi-Head Attention** The multi-head attention mechanism allows the model to focus on different parts of the sequence simultaneously. It is computed as follows:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W_O$$

where each head_i is calculated as:

$$\text{head}_i = \text{Attention}(QW_{Q_i}, KW_{K_i}, VW_{V_i})$$

and $W_{Q_i}, W_{K_i}, W_{V_i} \in \mathbb{R}^{d \times d_k}$, $W_O \in \mathbb{R}^{hd_k \times d}$.

- **Positional Encoding** Since the Transformer does not inherently capture the order of the sequence, positional encodings are added to the input embeddings:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right), \quad PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$

where pos is the position and i is the dimension index.

- **Layer Normalization and Residual Connections** Layer normalization and residual connections are applied to each sub-layer to stabilize and enhance the training process:

$$\text{LayerNorm}(x + \text{Sublayer}(x))$$

where the sub-layer could be either the multi-head attention or the feed-forward network.

- **Transformer Decoder Layer. Full Decoder-Only Transformer Model** A single decoder layer in the Transformer consists of a masked multi-head self-attention mechanism followed by a feed-forward neural network.

  The complete decoder-only Transformer model is built by stacking N identical decoder layers. The output of the final decoder layer is passed through a linear layer followed by a softmax function to generate the probability distribution over the target vocabulary:

  $$\text{Transformer}(X) = \text{softmax}(\text{Linear}(\text{Decoder}(X)))$$

  where $\text{Decoder}(X)$ represents the stacked decoder layers applied to the input sequence X.

2.2 Adversarial attacks against neural networks and the robustness verification problem

Given a neural network $\mathcal{N}$, the aim of adversarial attacks is to find a perturbation δ of an input x, such that x and $x+\delta$ are "similar" according to some domain knowledge, yet $\mathcal{O}(x) \neq \mathcal{O}(x+\delta)$. In this thesis, we use the common formulation of "similarity" in the field: two inputs are similar if the L_∞ norm of δ [1] is small. Under this formulation, finding an adversarial example can be defined as solving the following optimization problem:[2]

$$\min ||\delta||_\infty \text{ s.t } \mathcal{O}(x) \neq \mathcal{O}(x+\delta)$$

In practice, it is very hard to formally define "similar": should an image and a crop of it "similar"? Should two sentences differ by one synonym the same? We refer curious readers to the survey [141] for a comprehensive review of different formulations.

One natural defense against adversarial attacks, called *robustness verification*, is to prove that $\min ||\delta||_\infty$ must be greater than some user-specified threshold ϵ. Formally, given that $\mathcal{O}(x) = i$, we verify

$$\forall x' \in B(x,\epsilon) \cdot \forall j \neq i \cdot F(x')[i] - F(x')[j] > 0 \tag{2.1}$$

where $B(x,\epsilon)$ is a L_∞ norm-ball of radius ϵ centered at x: $B(x,\epsilon) = \{x' \mid ||x-x'||_\infty \leq \epsilon\}$. If Eq. (2.1) holds, we say that x is ϵ-robust.

[1] $L_\infty(\delta) = ||\delta||_\infty = \max(|\delta_0|, |\delta_1|, ...)$ where δ is the vector $[\delta_0, \delta_1, ...]$

[2] While there are alternative formulations of adversarial robustness (see [141]), in this document, we use adversarial attacks as a black box, thus, stating one formulation is sufficient.

2.3 SAT/SMT. Verifying neural networks as an SMT problem. Solving SAT/SMT.

The Boolean Satisfiability (SAT) problem Given a formula over Boolean variables and Boolean gates $\wedge$ (and), $\vee$ (or), $\neg$ (not), such as

$$(A \vee B) \wedge (C \vee D) \wedge \neg B$$

the SAT problem is to find the assignment for each of the variables (A, B, C and D in this example) so that the whole formula is evaluated to True. If such an assignment exists, we report SAT. If not, we report UNSAT. We introduce here some terminologies frequently used in SAT:

- A *literal* is either a variable or its negation, e.g $A, \neg B$
- A *clause* is a disjunction of literals, e.g a clause $c = \ell_0 \vee \ell_1 ... \vee \ell_k$
- A *unit clause* is a clause that contains exactly one literal
- A formula is in a *CNF* if it is a conjunction of clauses, e.g a CNF formula $F = c_0 \wedge c_1 \wedge ...c_i$

The Satisfiability modulo theories (SMT) problem While many problems in Computer Science and Mathematics can be encoded into SAT [54, 123], its expressiveness is still limited. SMT - Satisfiability modulo theories, is a natural extension of SAT in which Boolean variables could be extended to predicates in different theories. For example, an SMT formula in Quantifier-free Linear Real Arithmetic (QF-LRA):

$$\begin{aligned} &(x + 1 > 0 \vee x + y > 0) \\ \wedge &(x < 0 \ \vee x + y > 4) \\ \wedge &\neg(x + y > 0) \end{aligned}$$

The SMT problem is to find the assignment for each of the variables (x and y in this example) so that the whole formula is evaluated to True with respect to the theories.

Encoding DNNs and robustness properties as SMT formulas Any DNN can be encoded as an SMT formula. More specifically, the networks [3] considered in this thesis (feed-forward DNNs using ReLU activations) can be encoded as SMT formulas in QF-LRA. The encoding starts with creating a real variable for each variable in the input and the output. Then, for each layer, each neuron in the layer can be written as a constraint over its inputs.

- For a linear layer $z^i = \boldsymbol{W}^i h^{i-1} + \boldsymbol{b}^i$, the j^{th} variable in the output is encoded as

$$z^i_j = \sum_{k=1}^{len(h^{i-1})} \boldsymbol{W}^i_{k,j} * h^{i-1}_k + \boldsymbol{b}^i_j \tag{2.2}$$

- For a ReLU layer $h^i = \text{ReLU}(z^i)$, the j^{th} variable in the output is encoded as

$$h^i_j = \max(0, z^i_j) \tag{2.3}$$

  or

$$h^i_j = \text{if-then-else}(z^i_j > 0, z^i_j, 0) \tag{2.4}$$

The first 8 equations in Fig. 2.1b demonstrate how to encode XNET into SMT using this encoding.

More importantly, the specification of the robustness problem can also be encoded in QF-LRA: in Eq. (2.1), the neural network function F can be encoded in QF-LRA, the quantifier for x' can be written as linear bounds over each variable in the input, and since there are a fixed number of classes, the quantifier over i and j can be removed by checking Eq. (2.1) for each of the pair (i, j). Concretely, Fig. 2.1b shows how the neural network XNET and a verification specification can be encoded into a single SMT problem over QF-LRA.

Solving SAT/SMT We introduce here only the most relevant SAT/SMT terminologies and algorithms to the thesis. Since SAT/SMT are some of the most important and well-studied problems in computer science, it is impossible to survey all the work given the scope of this thesis. We refer curious readers to a more comprehensive study at [42].

[3] While in Chapter 6 we verify an explanation method for LLM, we do so by only looking at the task heads of the LLMs, which are feed-forward DNNs

Algorithm 1: The DPLL algorithm. Note that in the case of returning SAT, the assignment is not explicitly returned, but implied by the trace.

Input: A formula F in CNF
Output: `SAT` or `UNSAT`

while *there is a unit clause* c *in* F **do**
  $F \leftarrow F[c \leftarrow True]$;
if $F = True$ **then**
  return `SAT`;
for *variable* $v \in F$ **do**
  if $DPLL(F[v \leftarrow True]) = SAT$ **then**
    return `SAT`;
  if $DPLL(F[v \leftarrow False]) = SAT$ **then**
    return `SAT`;
return `UNSAT`;

Solving SAT with DPLL. Given a SAT problem, one naive way to decide its satisfiability is by trying every possible value and checking if the formula is evaluated to True. Given a formula of n variables, that means combing through all 2^n combinations, so this blind search approach doesn't scale very far. One of the earliest successful attempts to avoid doing a completely blind search is DPLL [33, 32].

Given a Boolean formula in CNF [4], DPLL alternates between two procedures: Boolean constant propagation (BCP), and search. During BCP, DPLL looks for all unit clauses and assigns the sole literals in them to True. This may change some other non-unit clauses into unit clauses. Consider the following example:

$$F = A \wedge (\neg A \vee B)$$

A is a unit clause, thus BCP assigns A to True. We have

$$\begin{aligned} F &= (\text{True}) \wedge (\neg \text{True} \vee B) \\ &= (\text{False} \vee B) \\ &= B \end{aligned}$$

At this point, we have another unit clause B.

[4] Fortunately, there are algorithms to convert an arbitrary Boolean formula to CNF, such as Tseytin transformation [129].

Given that BCP can create more unit clauses, DPLL runs BCP till a fixed-point, where no new unit clauses are introduced. At this point, DPLL employs the second procedure – search, which picks a variable and assigns a value to it. Together, the whole DPLL algorithm is summarized in Alg. 1

Algorithm 2: Typical CDCL algorithm

Input: A formula F in CNF, a trace ν
Output: `SAT` if F is satisfiable, `UNSAT` otherwise

```
1  if (BCP(F, ν) == CONFLICT) then
2  |  return UNSAT;
3  dl ← 0 ;                                                  // Decision level
4  while (not AllVariablesAssigned(F, ν)) do
5  |  (x, v) ← PickBranchingVariable(F, ν) ;                  // Decide stage
6  |  dl ← dl + 1 ;          // Increment decision level due to new decision
7  |  ν ← ν ∪ {(x, v)};
8  |  if (BCP(F, ν) == CONFLICT) then
9  |  |  β ← ConflictAnalysis(F, ν) ;                        // Conflict Analysis
10 |  |  if (β < 0) then
11 |  |  |  return UNSAT;
12 |  |  else
13 |  |  |  Backtrack(F, ν, β);
14 |  |  |  dl ← β ;        // Decrement decision level due to backtracking
15 return SAT;
```

Conflict-driven Clause Learning (CDCL) The original DPLL algorithm laid the foundation for SAT solving, but it had some limitations. CDCL [85] was developed to address these limitations by incorporating advanced techniques that significantly improve the performance of SAT solvers. While DPLL relies on backtracking when a conflict (contradiction) is encountered, CDCL improves upon this by analyzing conflicts when they occur, thus reducing the amount of redundant backtracking. It learns new clauses (conflict or learned clauses) that prevent the solver from repeating the same mistake. On top of that, CDCL uses Non-Chronological Backtracking (by using a trace ν to keep track of the made decisions), allowing the solver to jump back more than one level in the decision tree, directly to the point where the actual cause of the conflict originated. This results in fewer backtracks and faster convergence (compared to DPLL where the solver can only undo the most recent decision). Alg. 2 shows a typical CDCL algorithm.

Solving SMT with CDCL(T). To solve an SMT formula F, the most common

Algorithm 3: The CDCL(T) loop

Input : an SMT formula F
Output: `SAT` or `UNSAT`

```
1  clauseDB ← toCNF(F);
2  while True do
3      do
4          conflict ← BooleanPropagate(clauseDB);
5          changed ← False;
6          if conflict = ∅ then
7              conflict, changed ← theoryCheck() ;
8      while changed ∧ conflict = ∅;
9      if conflict ≠ ∅ then
10         level, lemma ← resolveConflict(conflict);
11         clauseDB ← clauseDB ∪ lemma;
12         if level < 0 then
13             return UNSAT;
14         backtrack(level) ;
15     else
16         if nextLiteral() = NULL then
17             return SAT ;
```

strategy (CDCL(T) [95]) is to solve the Boolean version of it first. Through a process called Boolean Abstraction [95], each unique theory predicate is abstracted by a Boolean variable, thus converting the original SMT formula into a Boolean formula. For example, consider the following SMT formula

$$F = (x > 0 \vee x < y) \wedge \neg(x > 0)$$

its Boolean abstracted formula is

$$F' = (A \vee B) \wedge \neg A$$

The mapping $\{x > 0 : A, x < y : B\}$ is called the *Theory Atom Map*, and its keys are called *Theory Atoms*. If the Boolean abstracted formula is UNSAT, it is safe to say that the original SMT problem is also UNSAT. If there exists an assignment I for the abstracted formula, at this point it is necessary to invoke the theory solvers to check if the assignment also satisfies the theory. If yes, we find an assignment for F. If not, the theory solvers

returns a reason why the current assignment is UNSAT through a learned clause c_T. The algorithm is summarized in Alg. 3.

2.4 Neural network verifiers

As discussed earlier, the neural network verification problem can be encoded as QF-LRA formulas, thus in theory can be solved by any off-the-shelf SMT solver such as Z3 [34] or CVC5 [9]. However, this naive approach doesn't scale beyond tiny networks. Thus, researchers have invented specialized tools to verify the robustness of neural networks. In this section, we look at the two major classes of neural network verifiers: constraint-based and abstraction-based. For a more comprehensive survey of existing verification algorithms and tools for neural networks, we refer curious readers to [81] and [3].

2.4.1 Constraint-based verifiers

SMT solvers usually support multiple theories (e.g. string, bitvector, etc.) as well as a combination of them, while neural network verifiers only need to reason about Quantifier-free Linear Real Arithmetic. Thus, dedicated neural network verifiers can exploit heuristics and architectures that may not applicable to other theories. To demonstrate this idea, we discuss one Constraint-based verifier – Marabou [70].

Marabou is a dedicated state-of-the-art NN verifier. Marabou extends the Simplex [94] algorithm for solving linear programming with special mechanisms to handle non-linear activation functions. Internally, Marabou encodes both the verification problem and the adversarial attacks as a system of linear constraints (the weighted sum and the properties) and non-linear constraints (the activation functions). For example, Fig. 2.1b shows how to encode the property that XNET is 0.04-robust at (0.06, 0.06). Same as Simplex, at each iteration, Marabou tries to fix a variable so that it doesn't violate its constraints. While in Simplex, a violation can only happen due to a variable becoming out-of-bound, in Marabou a violation can also happen when a variable doesn't satisfy its activation constraints.

By focusing only on neural networks with piecewise-linear activation functions, Marabou makes two insights: first, only a small subset of the activation nodes are relevant to the property under consideration. Hence, Marabou treats the non-linear constraints lazily and reduces the number of expensive case-splits, making it much faster than traditional SMT solvers. Second, Marabou repeatedly refines each variable's lower and upper bound,

hoping that many piecewise-linear constraints can be turned into linear (phase-fixed), reducing further the need for case splitting. Altogether, Marabou achieves state-of-the-art performance on a wide range of benchmarks [7].

2.4.2 Abstraction-based verifiers

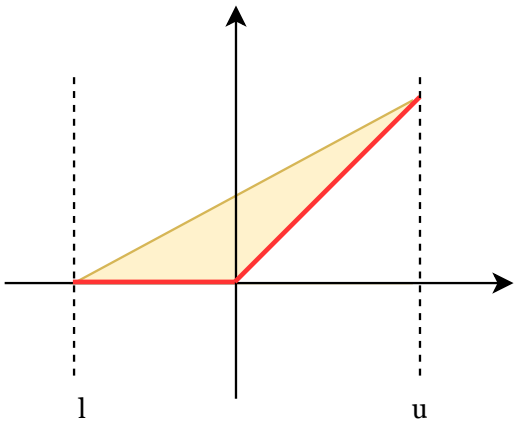
While Constraint-based verifiers such as Marabou can solve the encoded neural network verification precisely, their scalability remains an issue: at the end of the day, SMT problems are at best NP-complete (as hard as its Boolean abstracted SAT problem). Thus, ones may wish to make the problem easier by abstracting (over-approximating) the semantics of a DNN, in hope of claiming UNSAT faster. This approach is often known as Abstract Interpretation [28]. Concretely speaking, the aim of abstraction-based verifiers is to convert the original problem (which is NP-complete) to a Linear Programming problem (which can be solved in polynomial time) by abstracting away the source of non-linearity – the ReLU activation function. Then, given than $\mathcal{O}(x) = i$, abstraction-based verifiers solve the optimization problem

$$\forall j \neq i \cdot \min_{x' \in B(x,\epsilon)} F^*(x')[i] - F^*(x')[j] \tag{2.5}$$

in which F^* is the abstracted version of the neural network function F. If the minimum is greater than 0, it is safe to say that $F(x')[i] - F(x')[j] > 0$, making Eq. (2.1) true.

While often much faster than Constraint-based verifiers, Abstraction-based verifiers are not precise: if the computed minimum is smaller than 0, the verifier cannot conclude anything about the property. As with DPLL(T), at this point, we may need to add back some or all constraints from the original problem to check if the computed minimum is still satisfiable. This trade-off between being more precise and more efficient is the one that all abstraction-based verifiers have to balance. In this section, we survey some of the most prominent ones.

ERAN [121, 120] is an abstraction-based verifier that use polyhedron (concretely triangles) to approximate ReLU as well as other activation functions. Fig. 2.2 demonstrates this strategy. Given the same lower and upper bound to the input, this is also the tightest convex polyhedron that can be used to approximate the ReLU functions: the only way to be tighter is to "bend" the top face downwards, thus losing convexity. Recent extensions to ERAN include (but are not limited to) approximating multiple ReLUs at the same time [119, 92] and a GPU-based implementation [110].



(a) A ReLU (bold line) and its polyhedron (triangle) abstraction (shaded area)

$$x \leq u$$
$$x \geq l$$
$$y \geq 0$$
$$y \geq x$$
$$y \leq \frac{u(x-l)}{u-l}$$

(b) Approximating $y = \text{ReLU}(x)$ knowing the lower bound l and upper bound u of x

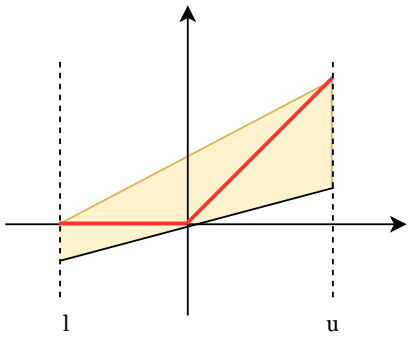
Figure 2.2: Approximating ReLUs using a polyhedron (triangle) in ERAN.

The Neural Network Verification (NNV) toolbox [127] , written in MATLAB, is designed for the formal analysis and verification of deep neural networks. NNV also uses polyhedron approximation but represents the abstracts using the star set representation [6]. This representation allows efficient over-approximation of the reachable sets, which is crucial for verifying neural network behavior.

α-Crown uses a 2-line abstraction for ReLU, as illustrated in Fig. 2.3. While being less tight than using triangles given the same lower and upper bounds, α-Crown makes the insight that now the bounds are functions of α, thus by optimizing α, α-Crown can derive tighter lower and upper bounds for each neuron. Recent extensions to α-Crown include (but are not limited to) α, β-Crown, which compute the exact minimum through branch-and-bound, making the method precise (at the cost of having the same worst-case complexity as solving QF-LRA).

2.5 Explanable Machine Learning

In contrast to traditional software in which both the code and the algorithms behind it are created by humans, DNNs are only designed by humans but their concrete parameters are



(a) A ReLU (bold line) and its linear abstraction (shaded area). The lower line has the equation $y = \alpha x$

$$x \leq u$$
$$x \geq l$$
$$y \geq \alpha x \text{ where } 0 < \alpha < 1$$
$$y \leq \frac{u(x-l)}{u-l}$$

(b) Approximating $y = \text{ReLU}(x)$ knowing the lower bound l and upper bound u of x

Figure 2.3: Linear approximation for ReLUs in α-CROWN.

learned from the data. Thus, DNNs' calculation process is often regarded as a "black box", and researchers have been trying to create tools and methods to explain what exactly is happening during a DNN's computation. We survey here some of the methods that are most relevant to the thesis and refer curious readers to [87] for a more detailed study.

In all of the surveyed methods, we use the same setup: given an input $x \in \mathbb{R}^{d_0}$, a model describes a function $F_N : \mathbb{R}^{d_0} \to \mathbb{R}^{d_L}$ where d_L is the number of classes in the classification problem (or the output dimension). An explanation method constructs an explanation map $E : \mathbb{R}^{d_0} \to \mathbb{R}^{d_0}$ that maps inputs to objects of the same dimensions.

2.5.1 Abductive explanations

Given an input $x \in \mathbb{R}^{d_0}$ and its predicted label $\mathcal{O}(x)$, an *abductive explanation*, or an *AXP* of x is a mapping $E_{AXP} : \mathbb{R}^{d_0} \to \{0,1\}^{d_0}$ s.t

$$\forall x' \in \mathbb{R}^{d_0} \cdot \bigwedge_{E_{AXP}(x)_i = 1} (x'_i = x_i) \implies \mathcal{O}(x') = \mathcal{O}(x) \tag{2.6}$$

Intuitively, the AXP constructs a subset of pixels of the original input ($E_{AXP}(x)_i = 1$) such that if those pixels are of exact values ($\forall x' \in \mathbb{R}^{d_0} \cdot x'_i = x_i$), then the picture must be of a certain class ($\mathcal{O}(x') = \mathcal{O}(x)$), no matter the values of the other pixels. One trivial AXP is the set of all features: if the value of all features is fixed, its output must be fixed as well. Hence, the research interest lies in finding the AXP of the smallest size.

2.5.2 Feature attribution (Saliency Map)

Feature attribution methods aim to compute a subset of the input's features that are most relevant to the final classification. For images, the features are pixels. For text, the features are words. Different methods differ by the exact formulation of "relevant". For images, the most commonly used formulation is the saliency map and its extensions.

Vanilla Gradient [117] (Saliency Maps). In this method, the pixels that are the most relevant are the ones with the highest gradients with respect to the predicted class. Concretely, we compute the gradient

$$E_{SM}(x) = \frac{\delta \mathcal{O}(x)}{\delta x} \tag{2.7}$$

The problem with Saliency Map lies in the fact that the gradient through ReLU is the same (0) for any input that is less than 0. To see why this is a problem, consider this toy neural net

$$y = 1 - ReLU(1 - x_0 - 2 * x_1)$$

equivalently

$$y = x_0 + 2 * x_1 \text{ if } x_0 + 2 * x_1 \leq 1, 1 \text{ otherwise.}$$

Suppose that we label y "good" if $y = 1$, "bad" otherwise. Given the input $x_0 = 0, x_1 = 0.6$, y is classified by the network as "good". However, the computed gradients with respect to y are 0 for both, telling us nothing about which of the two inputs contributes more to the fact that the predicted label is "good". This situation is known as "gradient saturation". Multiple works have been proposed to mitigate this: Gradient $\odot$ Input [115] computes

$$E_{prod}(x) = x \odot \frac{\delta \mathcal{O}(x)}{\delta x} \tag{2.8}$$

which leverages the sign and strength of the input, but doesn't completely overcome the gradient saturation problem. Smooth Grad [122] averages over saliency maps of noisy copies of an input by computing

$$E_{SG}(x) = \frac{1}{N} \sum_{i}^{N} E_{SM}(x + g_i) \tag{2.9}$$

in which g_i are noise vectors drawn i.i.d from a normal distribution. Similarly, Integrated Gradient [124] integrates the gradients as the inputs are scaled up from some baseline value $\bar{x}$ (eg: $\bar{x} = \vec{0}$) to their current value x, i.e computing

$$E_{IG}(x) = (x - \bar{x}) \int_0^1 \frac{\delta \mathcal{O}(\bar{x} + \alpha(x - \bar{x}))}{\delta(x)} d\alpha \tag{2.10}$$

Both Smooth Grad and Integrated Gradient address the gradient saturation problem [115], and are usually preferable to Vanilla Gradient.

2.5.3 Mechanical Interpretation of Large Language Models

Recent advancements in large language models (LLMs) have opened a fascinating array of capabilities. Originally designed for predicting the next token in a sequence, these networks have demonstrated proficiency in tasks that seemingly require a deep understanding of the underlying systems, such as solving logic puzzles [82] and generating working code snippets [104, 143]. Do these networks simply memorize conditional statistics, or do they implicitly construct internal representations of the process that generates the sequences they see? To answer this, researchers proposed a set of techniques, subsequently referred to as "mechanical interpretation", to investigate whether an LLM construct a real world model in its internal states.

Probing The first technique is to "probe" the internal representation of the network []. Let $\mathcal{N}$ be a neural network that takes in an input x and produces some internal representation $h_{\mathcal{N}}(x)$. Let $\mathcal{F}$ be a feature of x: e.g a chess board represented by a sequence of moves x, an AST corresponding to a code snippet x, etc. A *probe* $\mathcal{P}$ is a neural network that takes $h_{\mathcal{N}}(x)$ as an input, and outputs feature $\mathcal{F}(x)$ of x. If we can train $\mathcal{P}$ such that $\mathcal{P}(h_{\mathcal{N}}(x)) \approx \mathcal{F}(x)$, then we have more confidence that the internal representation $h_{\mathcal{N}}(x)$ encodes some forms of $\mathcal{F}$.

Intervention The second technique is to *corrupt*, or *intervene* the internal representation of the network. For this technique, we assume the existence of a highly accurate probe $\mathcal{P}$. We divide the neural network $\mathcal{N}$ into 2 parts: a *head* $\mathcal{H}$ and a *tail* $\mathcal{T}$, such that

$$\mathcal{H}(x) = h_{\mathcal{N}}(x) \tag{2.11}$$

$$\mathcal{T}(h_{\mathcal{N}}(x)) = \mathcal{N}(x) \tag{2.12}$$

For example, given a 10-layer LLM, the head can be the first 8 layers and the tail be the last 2 layers.

Let $\mathcal{N}(x)$ be the predicted output of the network. If for a large number of x, we can reliably find a perturbation vector δ s.t

$$\mathcal{T}(h_{\mathcal{N}}(x) + \delta) = \mathcal{N}(x') \tag{2.13}$$
$$\mathcal{P}(h_{\mathcal{N}}(x) + \delta) = \mathcal{F}(x') \tag{2.14}$$

then we can conclude that the representation space has a causal effect on the model. In words, that means if we can move in the representation space such that the probe and the network agree on the output, then we conclude about the causality.

2.6 Datasets and Target Networks

Throughout the document, we use the following datasets and pretrained networks accompanying them.

ACAS-Xu Modern aircrafts rely on automated collision avoidance systems to ensure their safe operation. Among them, one recent system known as Airborne Collision Avoidance System X - unmanned variant (ACAS Xu) [68], uses a large lookup table to map 7 sensor measurements to one of the 5 possible advisories (Table 2.1). This lookup table requires over 2GBs of memory, causing concerns for certified avionics hardware. One proposed solution is to use a much smaller DNN (in term of required memory) to replace the lookup table [68]. To improve the memory footprint even further, two of the features in the lookup tables are concretized, and the single proposed DNN is replaced by a set of 45 DNNs, corresponding to all 45 possible values of the feature combination. The original lookup table (120 millions datapoints) serves as the dataset for both training and testing the DNNs.

For ACAS-Xu, our verification targets are the 45 DNNs pretrained by [68]. They are all ReLU-activated Fully connected networks, with 6 layers and 300 ReLUs in total.

MNIST The *Modified National Institute of Standards and Technology database* [75] (MNIST) is a dataset of handwritten digits that is used for training digit recognition. The dataset contains 60,000 training images and 10,000 testing images. All inputs are grayscale images of size 28× 28 pixels, and outputs are one of the 9 possible digits.

Input features	Possible outputs
Distance from ownship to intruder	Clear-of-Conflict (COC)
Angle to intruder relative to ownship heading direction	Weak right
Heading angle of intruder relative to ownship heading direction	Strong right
Speed of ownship	Weak left
Speed of intruder	Strong left
Time until loss of vertical separation*	
Previous advisory (output)*	

Table 2.1: Inputs and outputs of the ACAS-Xu Systems. (*) The last 2 input features are concretized to the set of 45 possible values, hence the array of 45 DNNs.

For MNIST, we verify the set of the ReLU-activated Fully connected networks from VNNCOMP-2021 [7], with 256 ReLUs per layer and up to 6 layers.

CIFAR10 The CIFAR-10 dataset is a dataset of color images that is used for training image classification. The dataset has 60,000 images of 10 object classes, with exactly 6000 images per class. There are 50,000 training images and 10,000 test images. Inputs are of the size $3 \times 32 \times 32$: every image is of the size 32×32, and each pixel has 3 color channels (red, green, and blue). Unlike bigger image recognition datasets ([105, 78]), the classes are completely mutually exclusive, i.e an image can only be a truck or an automobile, but not both.

For CIFAR-10, we verify the set of ReLU-activated Convolutional neural networks from VNNCOMP-2021, each network has 2 convolutional layers followed by two fully connected feed-forward layers, with up to 10528 ReLUs in total.

OthelloGPT To study LLM and mechanical interpretation, we use the OthelloGPT model and the set of Othello games suggested in [77]. The board game Othello is played on an 8 by 8 board. Two players take turn to place white or black discs on the board. We use the two synthetic datasets used in [77]: a "legal" dataset in which 20 millions random games are recorded, and the model is tasked to predict the next legal move; a "championship" dataset collected from online sources, in which the model is tasked to predict the next best move. For both datasets, we use the pretrained 8-layer GPT models, both with an 8-head attention mechanism and a 512-dimensional hidden space.

Chapter 3

Towards Reliable Neural Specifications

3.1 Introduction

The advances in deep neural networks (DNNs) have brought a wide societal impact in many domains such as transportation, healthcare, finance, e-commerce, and education. This growing societal-scale impact has also raised some risks and concerns about errors in AI software, their susceptibility to cyber-attacks, and AI system safety [36]. Therefore, the challenge of verification and validation of AI systems, as well as, achieving trustworthy AI [138], has attracted much attention of the research community. Existing works approach this challenge by building on *formal methods* – a field of computer science and engineering that involves verifying properties of systems using rigorous mathematical specifications and proofs [137]. Having a formal specification — a precise, mathematical statement of what AI system is supposed to do is critical for formal verification. Most works [69, 70, 62, 60, 133] use the specification of adversarial robustness for classification tasks that states that the NN correctly classifies an image as a given adversarial label under perturbations with a specific norm (usually L_∞). Generally speaking, existing works use a paradigm of *data as specification* — the robustness of local neighborhoods of reference data points with ground-truth labels is the only specification of correct behaviors. However, from a learning perspective, this would lead to *overfitted* specification, since only local neighborhoods of reference inputs get certified.

As a refresher, Figure 1.2 illustrates this fundamental limitation of such overfitted specifications. Specifically, the test input Fig. 1.2a can hardly be verified even if all local

neighborhoods of all training images have been certified using the L_∞ norm. This is because adversarial examples like Fig. 1.2c lie in a much closer region compared to testing inputs (e.g., Fig. 1.2a), as a result, the truly verifiable region for a given reference input like Fig. 1.2b can only be smaller. All neural network verification approaches following such data-as-specification paradigm inherit this limitation *regardless* of their underlying verification techniques. In order to avoid such a limitation, a new paradigm for specifying what is correct or wrong is necessary. The intrinsic challenge is that manually giving a proper specification on the input space is no easier than directly programming a solution to the machine learning problem itself. We envision that a promising way to address this challenge is developing specifications directly on top of, instead of being agnostic to, the learned model.

We propose a new family of specifications, *neural representation as specification*, where neural activation patterns form specifications. The key observation is that inputs from the same class often share a neural activation pattern (NAP) – a carefully chosen subset of neurons that are expected to be activated (or not activated) for the majority of inputs in a class. Although two inputs are distant in a certain norm in the input space, the neural activations exhibited when the same prediction is made are very close. For instance, we can find a *single* NAP that is shared by *nearly all* training and testing images (including Fig. 1.2a and Fig. 1.2b) in the same class but not the adversarial example like Fig. 1.2c. We can further formally *verify* that *all possible* inputs following this particular NAP can never be misclassified. Specifications based on NAP enable successful verification of a broad region of inputs, which would not be possible if the data-as-specification paradigm were used. For the MNIST dataset, a verifiable NAP *mined* from the training images could cover up to 84% testing images, a significant improvement in contrast to 0% when using neighborhoods of training images as the specification. To our best knowledge, this is the first time that a significant fraction of *unseen* testing images have been formally verified.

This unique advantage of using NAPs as specification is enabled by the intrinsic information (or neural representation) embedded in the neural network model. Furthermore, such information is a simple byproduct of a prediction and can be collected easily and efficiently. Besides serving as reliable specifications for neural networks, we foresee other important applications of NAPs. For instance, verified NAPs may serve as proofs of correctness or certificates for predictions. We hope our initial findings shared in this chapter would inspire new interesting applications. We summarize our contribution as follows:

- We propose a new family of formal specifications for neural networks, *neural representation as specification*, which use activation patterns (NAPs) as specifications. We also introduce a tunable parameter to specify the level of abstraction of NAPs.

- We propose a simple yet effective approximate method to mine NAPs from neural networks and training datasets.
- We show that NAPs can be easily checked by out-of-the-box neural network verification tools used in VNNCOMP – the annual neural network verification competition, such as Marabou.
- We conduct thorough experimental evaluations from both statistical and formal verification perspectives. Particularly, we show that a single NAP is sufficient for certifying a significant fraction of unseen inputs.

3.2 Neural activation patterns

In this section, we discuss in detail neural activation patterns (NAPs), what we consider as NAPs and how to relax them, and what interesting properties of NAPs can be checked using neural network verification tools like Marabou [70].

3.2.1 NAPs and their relaxation

In our setting (Chapter 2), the output of each neuron is passed to the ReLU function before going to neurons of the next layer, i.e., $z^i(x) = \boldsymbol{W}^i\sigma(z^{i-1}(x)) + \boldsymbol{b}^i$. We abstract each neuron into two states: *activated* (if its output is positive) and *deactivated* (if its output is non-positive). Clearly, for any given input, each neuron can be either activated or deactivated.

Definition 3.2.1 (Neural Activation Pattern)**.** A *Neural Activation Pattern (NAP)* of a neural network is a tuple $\mathcal{P} := (A, D)$, where A and D are two disjoint subsets of activated and deactivated neurons, respectively.

Definition 3.2.2 (Partially ordered NAP)**.** For any given two NAPs $\bar{\mathcal{P}} := (\bar{A}, \bar{D})$ and $\mathcal{P} := (A, D)$. We say $\bar{\mathcal{P}}$ subsumes $\mathcal{P}$ iff A, D are subsets of $\bar{A}, \bar{D}$ respectively. Formally, this can be defined as:

$$\bar{\mathcal{P}} \preccurlyeq \mathcal{P} \iff \bar{A} \supseteq A \text{ and } \bar{D} \supseteq D \tag{3.1}$$

Moreover, two NAPs $\bar{\mathcal{P}}$ and $\mathcal{P}$ are equivalent if $\bar{\mathcal{P}} \preccurlyeq \mathcal{P}$ and $\mathcal{P} \preccurlyeq \bar{\mathcal{P}}$.

Definition 3.2.3 (NAP Extraction Function)**.** A NAP Extraction Function E takes a neural network $\mathcal{N}$ and an input x as parameters, and returns a NAP $\mathcal{P} := (A, D)$ where A and D represent all the activated and deactivated neurons of $\mathcal{N}$ respectively when passing x through $\mathcal{N}$.

With the above definitions in mind, we are able to describe the relationship between an input and a specific NAP. An input x *follows* a NAP $\mathcal{P}$ of a neural network $\mathcal{N}$ if:

$$E(\mathcal{N}, x) \preccurlyeq \mathcal{P} \tag{3.2}$$

For a given neural network $\mathcal{N}$ and an input x, it is possible x follows multiple NAPs. In addition, there are some trivial NAPs such as $(\emptyset, \emptyset)$ that can be followed by any input. From the representational learning point of view, these trivial NAPs are the least specific abstraction of inputs, which fails to represent data with different labels. Thus, we are prone to study more specific NAPs due to their rich representational power. Moreover, an ideal yet maybe impractical scenario is that all inputs with a specific label follow the same NAP. Given a label ℓ, and let S be the training dataset, and S_ℓ be the set of data labeled as ℓ, Formally, this scenario can be described as:

$$\forall x \in S_\ell \cdot \ E(\mathcal{N}, x) \preccurlyeq \mathcal{P}_\ell \iff \mathcal{O}(x) = \ell \tag{3.3}$$

This can be viewed as a condition for perfectly solving classification problems. In our view, $\mathcal{P}_\ell$, the NAP with respect to ℓ, if exists, can be seen as a certificate for the prediction of a neural network: inputs following $\mathcal{P}_\ell$ can be provably classified as ℓ by $\mathcal{N}$. However, in most cases, it is infeasible to have a perfect $\mathcal{P}_\ell$ that captures the exact inputs for a given class. On the one hand, there is no access to the ground truth of all possible inputs; on the other hand, DNNs are not guaranteed to precisely learn the ideal patterns. Thus, to accommodate standard classification settings in which Type I and Type II Errors are non-negligible, we relax $\mathcal{P}_\ell$ in such a way that only a portion of the input data with a specific label ℓ follows the relaxed NAP. The formal relaxation of NAPs is defined as follows.

Definition 3.2.4 (δ-relaxed NAP)**.** We introduce a relaxing factor $\delta \in [0, 1]$. We say a NAP is δ-relaxed with respect to the label ℓ, denoted as $\mathcal{P}_\ell^\delta := (A_\ell^\delta, D_\ell^\delta)$, if it satisfies the following condition:

$$\exists S'_\ell \subseteq S_\ell \ s.t. \ \frac{|S'_\ell|}{|S_\ell|} \geq \delta \ \textit{and} \ \forall x \in S'_\ell, \ E(\mathcal{N}, x) \preccurlyeq \mathcal{P}_\ell^\delta \tag{3.4}$$

Intuitively, the δ-relaxed factor controls the level of abstraction of NAP. When $\delta = 1.0$, not only $\mathcal{P}^{\delta=1.0}$ is the most precise (as all inputs from S_ℓ follow it) but also the least

Algorithm 4: NAP Mining Algorithm

Input : relaxing factor δ, neural network $\mathcal{N}$, dataset S_ℓ

Initialize a counter c_k for each neuron v_k;
for $x \in S_\ell$ **do**
    compute $E(\mathcal{N}, x)$;
    if v_k *is activated* **then**
        $c_k \mathrel{+}= 1$
$A_\ell \leftarrow \{v_k \mid \frac{c_k}{|S_\ell|} \geq \delta\}$;
$D_\ell \leftarrow \{v_k \mid \frac{c_k}{|S_\ell|} \leq 1 - \delta\}$;
$\mathcal{P}_\ell^\delta \leftarrow (A_\ell, D_\ell)$;

specific. In this sense, $\mathcal{P}^{\delta=1.0}$ can be viewed as the highest level of abstraction of the common neural representation of inputs with a specific label. However, being too abstract is also a sign of under-fitting, this may also enhance the likelihood of Type II Errors for NAPs. By decreasing δ, the likelihood of a neuron being chosen to form a NAP increases, making NAPs more specific. This may help alleviate Type II Errors, yet may also worsen the recall rate by producing more Type I Errors.

In order to effectively mine δ-relaxed NAPs, we propose a simple statistical method shown in Algorithm 4 [1]. Table 3.1 reports the effect of δ on the precision recall trade-off for mined δ-relaxed NAPs on the MNIST dataset. The table shows how many test images from a label ℓ follow $\mathcal{P}_\ell^\delta$, together with how many test images from other labels that also follow the same $\mathcal{P}_\ell^\delta$. For example, there are 980 images in the test set with label 0 (second column). Among them, 967 images follow $\mathcal{P}_{\ell=0}^{\delta=1.0}$. In addition to that, there are 20 images from the other 9 labels that also follow $\mathcal{P}_{\ell=0}^{\delta=1.0}$. With the decrease of δ, we can see that in both cases, both numbers decrease, suggesting that it is harder for an image to follow $\mathcal{P}_{\ell=0}^{\delta=.99}$ without being classified as 0 (the NAP is more precise), at the cost of having many images classified as 0 fail to follow $\mathcal{P}_{\ell=0}^{\delta=.99}$ (the NAP recalls worse). In short, the usefulness of NAPs largely depends on their precision-recall trade-off. Thus, choosing the right δ or the right level of abstraction becomes crucial in using NAPs as specifications in verification. We discuss this matter further in Section 3.3.

[1] Note that this algorithm is an approximate method for mining δ-relaxed NAP, whereas δ should be greater than 0.5, otherwise, $A_\ell^\delta \bigcap D_\ell^\delta \neq \emptyset$. We leave more precise algorithms for future work.

Table 3.1: The number of the test images in MNIST that follow a given δ.NAP. For a label i, $\bar{i}$ represents images with labels other than i yet follow δ.NAP^i. The leftmost column is the values of δ. The top row indicates how many images in the test set are of a label.

	0 (980)		1 (1135)		2 (1032)		3 (1010)		4 (982)		5 (892)		6 (958)		7 (1028)		8 (974)		9 (1009)	
	0	$\bar{0}$	1	$\bar{1}$	2	$\bar{2}$	3	$\bar{3}$	4	$\bar{4}$	5	$\bar{5}$	6	$\bar{6}$	7	$\bar{7}$	8	$\bar{8}$	9	$\bar{9}$
0.00	967	20	1124	8	997	22	980	13	959	25	874	32	937	26	1003	28	941	22	967	12
0.01	775	1	959	0	792	4	787	2	766	3	677	1	726	4	809	2	696	3	828	4
0.05	376	0	456	0	261	1	320	0	259	0	226	0	200	0	357	0	192	0	277	0
0.10	111	0	126	0	43	0	92	0	76	0	24	0	45	0	144	0	44	0	73	0

3.2.2 Interesting NAP properties

We expect that NAPs can serve as the key component in more reliable specifications of neural networks. As the first study on this topic, we introduce here three important ones.

The non-ambiguity property of NAPs We want our NAPs to give us some confidence about the predicted label of an input, thus a crucial sanity check is to verify that no input can follow two different NAPs of two different labels. Formally, we want to verify the following:

$$\forall x \cdot \forall i \neq j \cdot E(\mathcal{N}, x) \preccurlyeq \mathcal{P}_{\ell=i} \Longrightarrow E(\mathcal{N}, x) \not\preccurlyeq \mathcal{P}_{\ell=j} \tag{3.5}$$

Note that this property doesn't hold if either $A_{\ell=i} \bigcap D_{\ell=j}$ or $A_{\ell=j} \bigcap D_{\ell=i}$ is non-empty as a single input cannot activate and deactivate the same neuron. If that's not the case, we can encode and verify the property using verification tools.

NAP robustness property The intuition of using neural representation as specification not only accounts for the internal decision-making process of neural networks but also leverages the fact that NAPs themselves map to regions of our interests in the whole input space. In contrast to canonical ϵ-balls, these NAP-derived regions are more flexible in terms of size and shape. We explain this insight in more detail in Section 3.2.3. Concretely, we formalize this NAP robustness verification problem as follows: given a neural network $\mathcal{N}$ and a NAP $\mathcal{P}_{\ell=i}$, we want to check:

$$\forall x \in R \cdot \forall j \neq i \cdot F(x)[i] - F(x)[j] > 0 \tag{3.6}$$

in which

$$R = \{x \mid E(\mathcal{N}, x) \preccurlyeq \mathcal{P}_{\ell=i}\} \tag{3.7}$$

NAP-augmented robustness property Instead of only having the activation patterns as specification, we can still specify ϵ-balls in the input space for verification. This conjugated form of specification has two advantages: First, it focuses on the verification of valid test inputs instead of adversarial examples. Second, the constraints on NAPs are likely to make verification tasks effortless by refining the search space of the original verification problem, in most cases, allowing the verification on much larger ϵ-balls. We formalize the NAP-augmented robustness verification problem as follows: given a neural network $\mathcal{N}$, an input x, and a mined $\mathcal{P}_{\ell=i}$, we check:

$$\forall x' \in B^+(x, \epsilon, \mathcal{P}_{\ell=i}) \cdot \forall j \neq i \cdot F(x')[i] - F(x')[j] > 0 \tag{3.8}$$

in which $\mathcal{O}(x) = i$ and

$$B^+(x, \epsilon, \mathcal{P}_{\ell=i}) = \{x' \mid ||x - x'||_\infty \leq \epsilon, E(\mathcal{N}, x') \preccurlyeq \mathcal{P}_{\ell=i}\} \tag{3.9}$$

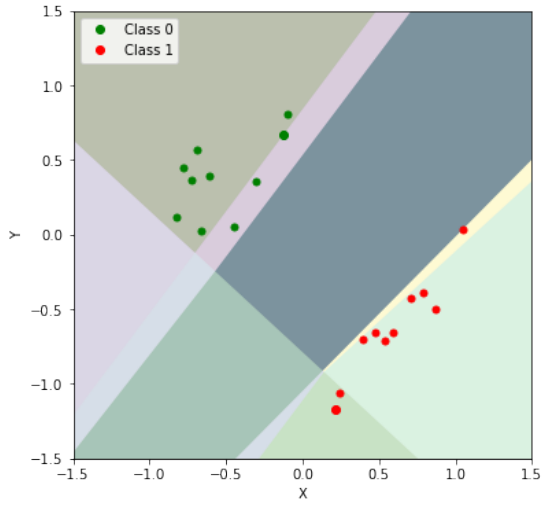
Working with NAPs using Marabou In this chapter, we use Marabou [70], a dedicated state-of-the-art NN verifier. Marabou extends the Simplex [94] algorithm for solving linear programming with special mechanisms to handle non-linear activation functions. Internally, Marabou encodes both the verification problem and the adversarial attacks as a system of linear constraints (the weighted sum and the properties) and non-linear constraints (the activation functions). Same as Simplex, at each iteration, Marabou tries to fix a variable so that it doesn't violate its constraints. While in Simplex, a violation can only happen due to a variable becoming out-of-bound, in Marabou a violation can also happen when a variable doesn't satisfy its activation constraints.

NAPs and NAP properties can be encoded using Marabou with little to no changes to Marabou itself. To force a neuron to be activated or deactivated, we add a constraint for its output. To improve performance, we infer ReLU's phases implied by the NAPs, and change the corresponding constraints[2].For example, given a ReLU $v_i = max(v_k, 0)$, to enforce v_k to be activated, we remove the constraint from Marabou and add two new ones: $v_i = v_k$, and $v_k \geq 0$.

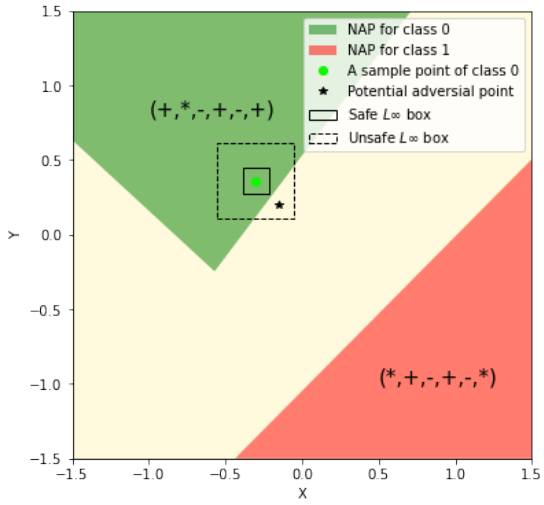
3.2.3 Case Study: Visualizing NAPs of a simple Neural Network

We show the advantages of NAPs as specifications using a simple example of a three-layer feed-forward neural network that predicts a class of 20 points located on a 2D plane. We trained a neural network consisting of six neurons that achieves 100% accuracy in the

[2]Marabou has a similar optimization, but the user cannot control when or if it is applied.



(a) Linear regions in different colors are determined by weights and biases of the neural network. Points colored either red or green constitute the training set.



(b) NAPs are more flexible than L_∞ norm-balls (boxes) in terms of covering verifiable regions.

Figure 3.1: Visualization of linear regions and NAPs as specifications compared to L_∞ norm-balls.

prediction task. The resulting linear regions as well as the training data are illustrated in Fig. 3.1a. Table 3.2 summarizes the frequency of states of each neuron based on the result of passing all input data through the network, and NAPs for labels 0 and 1. Fig. 3.1b visualizes NAPs for labels 0 and 1, and the unspecified region which provides no guarantees on data that fall into it. The green dot is so close to the boundary between $\mathcal{P}_{\ell=0}$ and the unspecified region that some L_∞ norm-balls (boxes) such as the one drawn in the dashed line may contain an adversarial example from the unspecified region. Thus, what we could verify ends up being a small box within $\mathcal{P}_{\ell=0}$. However, using $\mathcal{P}_{\ell=0}$ as a specification allows us to verify a much more flexible region than just boxes, as suggested by the NAP-augmented robustness property in Section 3.2.2. This idea generalizes beyond the simple 2D case, and we illustrate its effectiveness further with a critical evaluation in Section 3.3.3.

Table 3.2: The frequency of each ReLU and the dominant NAPs for each label. Activated and deactivated neurons are denoted by $+$ and $-$, respectively, and $*$ denotes an arbitrary neuron state.

Label	Neuron states	#samples	Dominant NAP
0 (Green)	$(+,-,-,+,-,+)$	8	$(+,*,-,+,-,+)$
	$(+,+,-,+,-,+)$	2	
1 (Red)	$(+,+,-,-,+,-)$	7	$(*,+,-,+,-,*)$
	$(-,+,-,-,+,-)$	2	
	$(+,+,-,-,+,+)$	1	

3.3 Evaluation

In this section, we validate our observation about the distance between inputs, as well as evaluate our NAPs and NAP properties on networks and datasets from VNNCOMP-21.

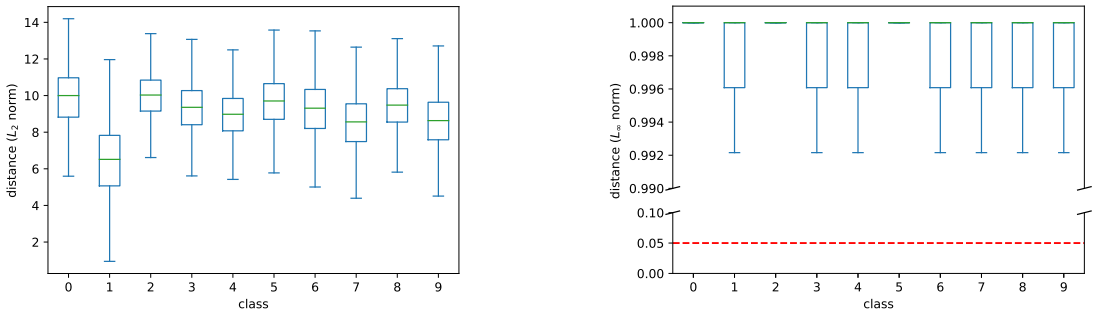
3.3.1 Experiment setup

Our experiments are based on benchmarks from VNNCOMP-21 [7] – the annual neural network verification competition. We use 2 of the datasets from the competition: MNIST and CIFAR10. For MNIST, we use the two largest models `mnistfc_256x4` and `mnistfc_256x6`, a 4- and 6-layers fully connected network with 256 neurons for each layer, respectively. For CIFAR10, we use the convolutional neural net `cifar10_small.onnx` with 2568 ReLUs. Experiments are done on a machine with an Intel(R) Xeon(R) CPU E5-2686 and 480GBs of RAM. Timeouts for MNIST and CIFAR10 are 10 and 30 minutes, respectively.

3.3.2 L_2 and L_∞ maximum verified bounds

We empirically find that the L_2 and L_∞ maximum verifiable bounds are much smaller than the distance between real data, as illustrated in Figure 1.2. We plot the distribution of distances in L_2 [3] and L_∞ norm between all pairs of images with the same label from the MNIST dataset, as shown in Figure 3.2. For each class, the smallest L_∞ distance of any two images is significantly larger than 0.05, which is the largest perturbation used in VNNCOMP-21.

[3]The L_2 metric is not commonly used by the neural network verification research community as it is less computationally efficient than the L_∞ metric.



(a) The distribution of L_2-norms between any two images from the same label. Images of digit (label) 1 are much similar than that of other digits.

(b) The distribution of L_∞-norms between any two images from the same label. The red line is drawing at 0.05 – the largest ϵ used in VNNCOMP-21

Figure 3.2: Distances between any two images from the same label (class) are quite significant under different metrics of norm.

This suggests that the *data as specification* paradigm (i.e., using reference inputs with perturbations bounded in L_2 or L_∞ norm) is not sufficient to verify test set inputs or unseen data. The differences between training and testing data of each class are usually significantly larger than the perturbations allowed in specifications using L_∞ norm-balls.

3.3.3 The NAP robustness property

We conduct two sets of experiments with MNIST and CIFAR10 to demonstrate the NAP and NAP-augmented robustness properties. The results are reported in Tables 3.3 to 3.5. For label ℓ from 0 to 9, 'Y' ('N') indicates that the network is (not) robust (i.e., no adversarial example of label ℓ exists). 'T/o' means the verification of robustness timed out.

MNIST with fully connected NNs In Figure 1.2, we show an illustrative image $\mathcal{I}$ (of digit 1) and its adversarial example within the distance of $L_\infty = 0.2$. As shown in Table 3.3, three different kinds of counter-example can be found within this distance. In contrast, the last row shows that all input images in the *entire input space* following the mined NAP specification $\mathcal{P}_{\ell=1}^{\delta=0.99}$ can be safely verified. It is worth noting that this specification covers 84% (959/1135) of the test set inputs (Table 3.1). To our best knowledge, this is the first specification for MNIST dataset that covers a substantial fraction of testing images. To some extent, it serves as a candidate machine-checkable definition of digit 1 in MNIST.

Table 3.3: Robustness of the illustrative example in Figure 1.2

	0	1	2	3	4	5	6	7	8	9
$\epsilon = 0.2$, no NAP	N	-	N	T/o	T/o	T/o	T/o	T/o	N	T/o
$\epsilon = 0.2, \delta = 1.0$	N	-	N	Y	T/o	T/o	Y	T/o	N	N
$\epsilon = 0.2, \delta = 0.99$	Y	-	Y	Y	Y	Y	Y	Y	Y	Y
no $\epsilon, \delta = 0.99$ (NAP robustness property)	Y	-	Y	Y	Y	Y	Y	Y	Y	Y

The second and third rows of Table 3.3 show the robustness of combining L_∞ norm perturbation and NAPs as the specification. The third row is well expected as the last row has shown that the network is robust against NAP itself (without L_∞ norm constraint). It is interesting to see that when we increase δ to 1.0, the mined NAP specification $\mathcal{P}_{\ell=1}^{\delta=1.0}$ becomes too general and covers a much larger region that includes more than 99% (1124/1135) testing images as shown in Table 3.1. As a result, together with $L_\infty = 0.2$ constraint, only two classes of adversarial examples can be safely verified, which is still better than only using $L_\infty = 0.2$ perturbation as the specification.

We further study how NAP-augmented specification helps to improve the verifiable bound. Specifically, we collect all (x, ϵ) tuples in VNNCOMP-21 MNIST benchmarks that are known to be not robust (an adv. example is found in $B(x, \epsilon)$). Among them, the first six tuples correspond to `mnistfc_256x4` and the last one corresponds to `mnistfc_256x6`. Table 3.5 reports the verification results with NAP augmented specification.

For the first six instances, using the NAP augmented specifications $B^+(\cdot, \epsilon = 0.05, \mathcal{P}^{\delta=1.0})$ enables the verification against more labels, outperforming using only L_∞ perturbation as the specification. By slightly relaxing the NAP ($\delta = 0.99$) , *all* of the chosen inputs can be proven to be robust. Furthermore, with $\delta = 0.99$, we can verify the robustness for 6 of the 7 inputs (Table 3.5) with $\epsilon = 0.3$, which is *an order of magnitude* bigger bound than before. Note that decreasing δ specifies a smaller region, usually allowing verification with bigger ϵ, but a smaller region tends to cover fewer testing inputs. Thus, choosing an appropriate δ is crucial for having useful NAPs.

CIFAR10 with CNN To show that our insights and methods can be applied to more complicated datasets and network topologies, we conduct the second set of experiments using convolutional neural nets trained on the CIFAR10 dataset. We extract all (x, ϵ) tuples in the CIFAR10 dataset that are known to be not robust from VNNCOMP-21 (an

Table 3.4: Augmented robustness with CIFAR10 and CNN.

ϵ	0.012			0.024			0.12		
δ	0.99	0.95	0.9	0.99	0.95	0.9	0.99	0.95	0.9
$\mathcal{O}(x_0) = 8$	Y	Y	Y	N	T/o	Y	T/o	Y	Y
$\mathcal{O}(x_1) = 6$	T/o	N	Y	N	N	Y	N	N	Y
$\mathcal{O}(x_2) = 0$	Y	Y	Y	Y	Y	Y	N	N	N
$\mathcal{O}(x_3) = 1$	N	N	N	N	N	N	N	N	N
$\mathcal{O}(x_4) = 9$	N	Y	Y	N	N	N	N	N	N
$\mathcal{O}(x_5) = 7$	Y	Y	Y	N	T/o	Y	N	Y	Y
$\mathcal{O}(x_6) = 3$	Y	Y	Y	Y	Y	Y	N	N	N

adv. example is found in $B(x, \epsilon)$) and verify them using augmented NAP. For CIFAR10, $\mathcal{P}^{\delta=1.0}$ does not exist, thus we use $\mathcal{P}^{\delta=.99}$, $\mathcal{P}^{\delta=.95}$ and $\mathcal{P}^{\delta=.90}$. We follow the scenario used in VNNCOMP-21 and test the robustness against $(correctLabel + 1) \mod 10$. The results are reported in Table 3.4. As with MNIST, we observe that by relaxing δ, we were able to verify more examples at every ϵ. Even with $\epsilon = 0.12$ ($10\times$ the verifiable bound, which translates to an input space $10^{3072}\times$ bigger!), by slightly relaxing δ to 0.9, we can verify 3 out of 7 inputs.

3.3.4 The non-ambiguity property of mined NAPs

We evaluate the non-ambiguity property of our mined NAP at different δs on MNIST. At $\delta = 1.0$, we can construct inputs that follow any pair of NAP, indicating that $\mathcal{P}^{\delta=1.0}$s do not satisfy the property. However, by setting $\delta = 0.99$, we are able to prove the non-ambiguity for *all* pairs of NAPs, through both trivial cases and invoking Marabou. This is because relaxing δ leaves more neurons in NAPs, making it more difficult to violate the non-ambiguity property.

The non-ambiguity property of NAPs holds an important prerequisite for neural networks to achieve a sound classification result. Otherwise, the final prediction of inputs with two different labels may become indistinguishable. We argue that mined NAPs should demonstrate strong non-ambiguity properties and ideally, all inputs with the same label i should follow the same $\mathcal{P}_{\ell=i}$. However, this strong statement may fail even for an accurate model when the training dataset itself is problematic, as what we observed in Fig. 1.2d. These examples are not only similar to the model but also to humans despite being labeled

differently. The experiential results also suggest our mined NAPs do satisfy the strong statement proposed above if excluding these noisy samples.

3.4 Related work and Future Directions

Abstract Interpretation in verifying Neural Networks The software verification problem is undecidable in general [102]. Given that a Neural Network can also be considered a program, verifying any non-trivial property of a Neural network is also undecidable. Prior work on neural network verification includes specifications that are linear functions of the output of the network: Abstract Interpretation (AbsInt) [28] pioneered a happy middle ground: by sacrificing completeness, an AbsInt verifier can find proof much quicker, by over-approximating reachable states of the program. Many NN-verifiers have adopted the same technique, such as DeepPoly [121], CROWN [133], NNV [128], etc. They all share the same insight: the biggest bottleneck in verifying Neural Networks is the non-linear activation functions. By abstracting the activation into linear functions as much as possible, the verification can be many orders of magnitude faster than complete methods such as Marabou. However, there is no free lunch: Abstract-based verifiers are inconclusive and may not be able to verify properties even when they are correct.[4] On the other hand, the *neural representation as specification* paradigm proposed in this work can be naturally viewed as a method of Abstract Interpretation, in which we abstract the state of each neuron to only activated and deactivated by leveraging NAPs. We would like to explore more refined abstractions such as $\{(-\infty, 0], (0, 1], (1, +\infty)\}$ in future work.

Neural Activation Pattern in interpreting Neural Networks There are many attempts aimed to address the black-box nature of neural networks by highlighting important features in the input, such as Saliency Maps [117, 109] and LIME[101]. But these methods still pose the question of whether the prediction and explanation can be trusted or even verified. Another direction is to consider the internal decision-making process of neural networks such as Neural Activation Patterns (NAP). One popular line of research relating to NAPs is to leverage them in feature visualization [144, 21, 39], which investigates what kind of input images could activate certain neurons in the model. Those methods also have the ability to visualize the internal working mechanism of the model to help with transparency. This line of methods is known as activation maximization. While being great at explaining the prediction of a given input, activation maximization methods

[4] Methods such as alpha-beta CROWN [133] claim to be complete even when they are Abstract-based because the abstraction can be controlled to be as precise as the original activation function, thus reducing the method back to a complete one.

do not provide a specification based on the activation pattern: at best they can establish a correlation between seeing a pattern and observing an output, but not causality. Moreover, moving from reference sample to revealing neural network activation pattern is limiting as the portion of NAP uncovered is dependent on the input data. This means that it might not be able to handle cases of unexpected test data. Conversely, our method starts from the bottom up: from the activation pattern, we uncover what region of input can be verified. This property of our method grants the capability to be generalized. Motivated by our promising results, we would like to generalize our approach to modern deep learning models such as Transformers [130], which employ much more complex network structures than a simple feed-forward structure.

3.5 Conclusion

We propose a new paradigm of neural network specifications, which we call *neural representation as specification*, as opposed to the traditional *data as specifications*. Specifically, we leverage neural network activation patterns (NAPs) to specify the correct behaviours of neural networks. We argue this could address two major drawbacks of "data as specifications". First, NAPs incorporate intrinsic properties of networks which data fails to do. Second, NAPs could cover much larger and more flexible regions compared to L_∞ norm-balls centred around reference points, making them appealing to real-world applications. We also propose a simple method to mine relaxed dominant NAPs and show that working with NAPs can be easily supported by modern neural network verifiers such as Marabou. Through a simple case study and thorough valuation on the MNIST dataset, we show that using NAPs as specifications not only address major drawbacks of *data as specifications*, but also demonstrate important properties such as no-ambiguity and one order of magnitude stronger verifiable bounds. We foresee verified NAPs have the great potential of serving as simple, reliable, and efficient certificates for neural network predictions.

This chapter is adapted from the following published work:

- Nham Le, Chuqin Geng, Xiaojie Xu, Zhaoyue Wang, Arie Gurfinkel, Xujie Si. Towards Reliable Neural Specifications. In Proceedings of the 40 th International Conference on Machine Learning. 2023, Honolulu, Hawaii, USA.

Table 3.5: Inputs that are not robust can be augmented with a NAP to be robust. With $\delta = 0.99$, all inputs can be verified to be robust at $\epsilon = 0.05$ – the largest checked ϵ in VNNCOMP-21(not shown)

	0	1	2	3	4	5	6	7	8	9
$\mathcal{O}(\mathbf{x_0}) = \mathbf{0}$										
$\epsilon = 0.05, \delta = 1.0$	-	Y	Y	Y	Y	Y	Y	Y	Y	Y
$\epsilon = 0.3, \delta = 0.99$	-	Y	Y	Y	Y	Y	Y	Y	Y	Y
$\mathcal{O}(\mathbf{x_1}) = \mathbf{1}$										
$\epsilon = 0.05, \delta = 1.0$	Y	-	Y	Y	Y	Y	Y	Y	N	Y
$\epsilon = 0.3, \delta = 0.99$	Y	-	Y	Y	Y	Y	Y	Y	Y	Y
$\mathcal{O}(\mathbf{x_2}) = \mathbf{0}$										
$\epsilon = 0.05, \delta = 1.0$	-	T/o	T/o	Y	T/o	T/o	Y	N	T/o	T/o
$\epsilon = 0.3, \delta = 0.99$	-	Y	Y	Y	Y	Y	Y	Y	Y	Y
$\mathcal{O}(\mathbf{x_3}) = \mathbf{7}$										
$\epsilon = 0.05, \delta = 1.0$	N	T/o	Y	Y	T/o	T/o	Y	-	N	T/o
$\epsilon = 0.3, \delta = 0.99$	Y	Y	Y	Y	Y	Y	Y	-	Y	Y
$\mathcal{O}(\mathbf{x_4}) = \mathbf{9}$										
$\epsilon = 0.05, \delta = 1.0$	T/o	Y	Y	Y	Y	Y	N	Y	N	-
$\epsilon = 0.3, \delta = 0.99$	Y	T/o	T/o	Y	N	Y	T/o	T/o	T/o	-
$\mathcal{O}(\mathbf{x_5}) = \mathbf{1}$										
$\epsilon = 0.05, \delta = 1.0$	Y	-	N	Y	Y	Y	Y	N	N	N
$\epsilon = 0.3, \delta = 0.99$	Y	-	Y	Y	Y	Y	Y	Y	Y	Y
$\mathcal{O}(\mathbf{x_6}) = \mathbf{9}$										
$\epsilon = 0.05, \delta = 1.0$	T/o	T/o	T/o	T/o	T/o	T/o	T/o	T/o	T/o	-
$\epsilon = 0.3, \delta = 0.99$	Y	Y	Y	Y	Y	Y	Y	Y	Y	-
(`mnistfc_256x6`)										

Chapter 4

cvc5-d: Towards a distributed SMT solver

As discussed in Chapter 2, at the heart of all constraint-based neural network verifiers lies an SMT solver. In this chapter, we introduce CVC5-D, our portfolio-based distributed SMT solver. We propose a general architecture consisting of two main components: (i) solvers extended with the capability of sharing and importing information on the fly while solving; and (ii) a central manager that orchestrates and monitors solvers while also deciding which information to share with which solvers. We introduce new information-sharing strategies based on the idea of maximizing the amount of "good" diversity in the system. We show that on hard benchmarks from recent related work, CVC5-D instantiated with the cvc5 SMT solver outperforms a state-of-the-art partitioning-based approach, is competitive with existing portfolio approaches, and enables portfolio solving for new benchmarks.

4.1 Introduction

Solvers for satisfiability modulo theories (SMT) are used as general-purpose constraint solvers in a wide variety of applications, including those arising in computer science [16, 49], mathematics [54, 123], operations research [108], and more. Unsurprisingly, as users push SMT solvers to solve more diverse and challenging problems, solver performance becomes the limiting factor in many applications.

Today, state-of-the-art SMT solvers like cvc5 [9], Yices [37], and Z3 [34], do not scale horizontally to solve hard instances faster, and if the solving job times out or crashes, any

work done during the solving attempt is lost.An effective strategy for distributed SMT solving could address both issues: it can help scale SMT solving across multiple threads and machines, and by sharing information among solver instances, any progress made can be retained and used by others, even if one of the instances crashes or fails.

Two main approaches to distributed SMT solving have been explored: portfolio solving and divide-and-conquer. Portfolio solving is essentially a race between multiple independent SMT solver instances. Each solver is different in some way: either it is a completely different solver, or it is configured differently, or it is provided with a different (but logically equivalent) input. Portfolio solving aims to leverage the well-known high variance that often exists when solving equivalent SMT problems: the hope is that one of the solvers in the portfolio finishes quickly. Portfolio solving can be enhanced by sharing information among the solver instances. Typically, this information consists of formulas that the SMT solvers have learned that can be used to prune the search space. In divide-and-conquer solving, a single problem is partitioned in such a way that if each partition is solved, this provides a solution to the original problem. The main challenge is finding a way to divide the problem that actually improves performance.

In this chapter, we introduce CVC5-D, a new tool for portfolio-based distributed SMT solving. CVC5-D's architecture consists of two main components: (i) solvers extended with the capability to share and import information on the fly while solving; and (ii) a central manager that orchestrates and monitors solvers while also deciding which information to share with which solvers. We also introduce a new information-sharing strategy based on the idea of maximizing the amount of "good" diversity in the system. On hard benchmarks from recent work [136], CVC5-D outperforms state-of-the-art partitioning-based distributed cvc5 [136] and is competitive with existing portfolio approaches with information sharing [84]. We also show that CVC5-D provides a significant speed-up on string benchmarks, an important category of benchmarks that has not previously been attempted with distributed solvers.

In summary, our contributions include:

- a flexible and general architecture for portfolio-based SMT solving with information sharing;
- new portfolio strategies including *delayed sharing* and *guided randomization*;
- an implementation in CVC5-D; and
- an evaluation of CVC5-D and existing systems on several sets of challenging benchmarks.

The rest of the chapter is organized as follows. Section 4.2 covers background and related work. Section 4.3 describes the architecture of CVC5-D. Section 4.4 explains our novel portfolio strategies, and Section 4.5 provides additional implementation details. Experimental results are reported in Section 4.6, and Section 4.7 concludes.

4.2 Preliminaries

We assume the standard many-sorted first-order logic setting with the usual notions of terms, interpretations, and theories. We assume a fixed background theory $\mathcal{T}$ (which could be a composition of one or more individual theories). An *atom* is a term of sort BOOL that does not contain any proper sub-terms of sort BOOL. A *literal* is either an atom or the negation of an atom. A clause is a disjunction of literals, and a *cube* is a conjunction of literals. A formula is a term of sort BOOL and is *satisfiable* (resp., *unsatisfiable*) if it is satisfied by some (resp., no) $\mathcal{T}$-interpretation. A formula whose negation is unsatisfiable is *valid*.

4.2.1 CDCL(T)-based SMT solvers

Most modern SMT solvers are based on the CDCL(T) framework [95], in which a SAT solver and one or more theory solvers cooperate. The SAT solver incrementally builds a truth assignment for the *Boolean skeleton* of the formula, obtained by replacing each unique atom by a Boolean variable. It does this using a standard CDCL loop that is modified to also take into account theory reasoning. The modified CDCL($\mathcal{T}$) approach is shown in Alg. 5. Initially, an input formula F is converted to conjunctive normal form (CNF), and each clause is stored in a clause database. The main loop first calls Boolean propagation, which may assign some atoms to true or false. If Boolean propagation produces no conflicts, then the theory solvers are called to check for theory conflicts. These two steps repeat until a fixed point is reached. If there is a conflict, it is resolved by learning a conflict lemma and backtracking to an earlier level in which there is no conflict. Otherwise, the *nextLiteral* function is used to make a case split on a new literal. More details can be found in [12].

4.2.2 Portfolio solving with lemma sharing

SMT solvers are highly sensitive. Small changes to the input formula or solver heuristics can result in orders of magnitude difference in solving time [50]. While a cause of frustration for

Algorithm 5: The CDCL(T) loop

```
   Input  : an SMT formula F
   Output: SAT or UNSAT
1  clauseDB ← toCNF(F);
2  while True do
3  |  do
4  |  |  conflict ← BooleanPropagate(clauseDB);
5  |  |  changed ← False;
6  |  |  if conflict = ∅ then
7  |  |  |  conflict, changed ← theoryCheck() ;
8  |  while changed ∧ conflict = ∅;
9  |  if conflict ≠ ∅ then
10 |  |  level, lemma ← resolveConflict(conflict);
11 |  |  clauseDB ← clauseDB ∪ lemma;
12 |  |  if level < 0 then
13 |  |  |  return UNSAT;
14 |  |  backtrack(level) ;
15 |  else
16 |  |  if nextLiteral() = NULL then
17 |  |  |  return SAT ;
```

users, this phenomenon can be leveraged to create an effective *portfolio* solving strategy: multiple solvers (each configured differently or with permuted, but logically equivalent, inputs) are run in a "racing" mode and the result of the fastest one is returned. This approach has been explored extensively for both SAT and SMT solving [8, 83, 84, 142] and produces reliable speed-ups [139]. Still, portfolio solving is limited by the performance of the best and luckiest individual solver, leading to diminishing returns with increasing parallelism. Additional perfomrance can be obtained with *information sharing*. Each solver in the portfolio shares its learned conflict lemmas with the others, with the hope that this exchange of information will help find the solution faster.

Implementing a lemma-sharing portfolio in practice is highly non-trivial. System-wise, one must provide scalability, fault tolerance, and low overhead; algorithmic-wise, one must find a good balance between sharing useful information and overloading the system with too many lemmas. Moreover, a well-designed distributed solver should be modular and general, leaving room for future extensions. Ideally, it should also accommodate a wide range of different solvers, support new sharing strategies, and be compatible with other

parallel strategies such as partitioning. After a review of related work, we discuss our design and implementation, including design decisions that aim to meet the criteria mentioned above.

4.2.3 Related Work

Parallel strategies for SAT solving have been explored extensively [8, 55, 63, 142]. SMT solvers must take into account the more sophisticated CDCL(T) architecture and the different performance profiles of SMT applications. However, the two main approaches for parallel SAT solving are also found in the existing research literature on parallel SMT solving, namely *portfolio solving* and *partitioning*.

Portfolio solving for SMT. Z3 was the first SMT solver to implement portfolio solving with information sharing [139]. The Z3 implementation focuses on a shared-memory implementation and achieves a speed-up of 3.5x on average for moderately difficult integer difference logic benchmarks using a portfolio of four copies of Z3. The sharing strategy used is simple: lemmas with eight literals or fewer are shared, and others are not. Shared lemmas are put into a queue, and each solver in the portfolio checks its queue whenever it backtracks to decision level 0. Unfortunately, portfolio solving is no longer supported in recent versions of Z3.

SMTS [83] is another system implementing portfolio solving with information sharing. As with the Z3 approach, lemmas to be shared are loaded into queues that are accessed when the solvers backtrack to decision level 0. SMTS uses a central database to store shared lemmas. A filtering heuristic is used to decide which lemmas to add to the database, and a selection heuristic is used to decide which lemmas to share from the database. SMTS obtains its best results using a filter that discards lemmas with more than four literals and a selection heuristic that randomly samples from the database. The SMTS authors specifically flag the need for better filtering and selection techniques in their discussion of future work. Our work builds on and extends these previous approaches in several ways, as we discuss in the next section.

Partitioning in SMT. SMTS [83] implements several partitioning strategies that outperform sequential solving. Relatedly, Wilson et al. [136] implement a partitioning-based parallel solver using cvc5 (which we will refer to as CVC5-P going forward) and show that it outperforms traditional portfolio solving on a set of challenging benchmarks. CVC5-P does not use any information sharing, leaving the integration of sharing to future work. SMTS does explore a limited form of sharing mixed with partitioning: each partition can be solved using a portfolio with lemma sharing, which yields even better performance.

The focus of this chapter is on portfolio solving with sharing but without partitioning. We aim to build a robust and high-performance solution that could be expanded to include partitioning strategies in future work.

4.3 An Architecture for Portfolio-Based SMT Solving

In this section, we describe a general architecture for portfolio-based SMT solving and contrast it with prior approaches. Figure 4.1 depicts our architecture. It is designed to run on either a cluster of computing nodes or a multicore machine. Multiple solver instances (called *workers*) work on the same problem and share information through a central manager (or *broker*). The workers are SMT solvers instrumented to be able to export and import learned lemmas on the fly. Workers also track local statistics about lemma imports, exports, and filtering.

The central manager plays two roles. First, in the *control plane* (Fig. 4.1a), it manages the system by starting, configuring, monitoring, and terminating workers, and by monitoring the overall system (through telemetry collected at each solver) and network health (through periodically transmitted ping/pong messages). Second, in the *data plane* (Fig. 4.1b), it controls system data flow by managing lemma exchange between workers and by tracking and monitoring solver and system-level lemma statistics. In particular, the data-plane manager (i) tracks which lemmas arrive from which individual workers and (ii) decides which lemmas to forward to which workers. This already enables a finer level of control than in previous approaches, where lemma sources are not tracked and static selection criteria are used to decide which lemmas to share. The broker tracks system telemetry for both control and data, including statistics such as the number of lemmas exported or imported so far, time spent in various phases of processing those lemmas, whether a worker has solved its copy of the problem, and so forth.

We advocate a simple hub-and-spoke architecture, similar to that used in SMTS [84]. Using a central broker simplifies coordination and does not require workers to synchronize with each other We have also observed empirically that in our implementation, the broker is not a communication bottleneck (see Sec. 4.6). Our hub-and-spoke architecture tolerates worker failure and communication lag or failure. The design makes progress as long as the central manager and some workers are active. The manager is a single point of failure, but can be engineered to be robust.

4.3.1 Workers

As mentioned above, the workers are SMT solvers modified to support importing and exporting of learned lemmas during search. This allows for more fine-grained information sharing than prior approaches, where lemmas are only imported at decision level 0, and requires modifying the CDCL loop as shown in Alg. 6. The loop now calls an export procedure whenever a new lemma is learned as a result of conflict analysis (Alg. 6). Additionally, during the propagation phase, the worker adds lemmas received from the broker to its database by invoking an import procedure (Alg. 6). The worker sends telemetry to the broker whenever lemmas are exported or imported (Alg. 6 and Alg. 6). Each solver has a mechanism for locally filtering lemmas. The goal is to import and export only *useful* lemmas. We discuss various considerations for local filtering in Section 4.5.

4.3.2 Central Manager

The central manager (broker) sets up and configures both the workers and network communication channels and manages both the control and data planes. During solving, it coordinates the exchange of information between workers and detects termination.

A major role of the central manager is to distribute lemmas learned by one worker to the other workers, while discarding duplicates and managing additional filters. Because multiple workers can learn and export identical lemmas, the broker ensures that each unique lemma is only forwarded (at most) once to each worker. Again, this offers a more fine-grained control mechanism than prior work, in which all lemmas up to a certain size are always shared (Z3) or lemmas are sampled randomly (SMTS) from the database of all shared lemmas.

The core broker algorithm is shown in Alg. 7. The broker maintains two global variables: *archivedLemmas* is the set of all lemmas it has received; and *lemmaSolverMap* is a map from lemmas to worker ids that keeps track of the origin(s) of each lemma. When the broker receives a lemma, the lemma is canonicalized by sorting the set of its literals (Alg. 7). This ensures that one source of lemma redundancy is eliminated. The broker then uses this canonical form to detect whether the lemma is new (i.e., not in *archivedLemmas*) and to update the map *lemmaSolverMap*. Function *shouldSend* controls the timing of when lemmas are transmitted to the workers. When *shouldSend* is true, the broker sends each lemma l stored in *lemmaSolverMap* to the workers that did not export it. We discuss implementation choices for *shouldSend* in Section 4.5.

Algorithm 6: Modified CDCL(T) loop with lemma sharing

Input : an SMT formula F
Output: SAT or UNSAT

```
1  clauseDB ← toCNF(F) ;
2  while True do
3      do
4          conflict ← BooleanPropagate(clauseDB);
5          changed ← False;
6          if conflict = ∅ then
7              newLemmas ← importLemmas();
8              clauseDB ← clauseDB ∪ newLemmas;
9              sendtelemetry();
10             conflict, changed ← theoryCheck() ;
11     while (newLemmas ≠ ∅ ∨ changed) ∧ conflict = ∅;
12     if conflict ≠ ∅ then
13         level, lemma ← resolveConflict(conflict);
14         exportLemma(lemma);
15         sendtelemetry();
16         clauseDB ← clauseDB ∪ lemma;
17         if level < 0 then
18             return UNSAT;
19         backtrack(level) ;
20     else
21         if nextLiteral() = NULL then
22             return SAT ;
```

Algorithm 7: The broker's core lemma exchange routine

```
1  archivedLemmas ← ∅ ;
2  lemmaSolverMap ← ∅ ;
3  while True do
4  |  ℓ, w ← readMessage();
5  |  ℓ ← canonicalize(ℓ);
6  |  if ℓ ∈ archivedLemmas then
7  |  |  continue;
8  |  else
9  |  |  lemmaSolverMap[ℓ].add(w);
10 |  if shouldSend() then
11 |  |  for ℓ ∈ lemmaSolverMap do
12 |  |  |  send(ℓ, allWorkers − lemmaSolverMap[ℓ]) ;
13 |  |  |  lemmaSolverMap.pop(ℓ) ;
14 |  |  |  archivedLemmas.add(ℓ);
```

4.4 Portfolio Strategies

Constructing effective strategies for portfolio solving with information sharing requires balancing trade-offs from a number of different goals:

- *Maximize diversity:* workers should work on different parts of the search space to avoid redundant work.
- *Share useful lemmas:* ideally, workers should export lemmas that are useful to all instances. A common heuristic for evaluating the value of a lemma is its size (i.e., number of literals in the clause). Smaller clauses are more likely to be useful, as they prune a larger portion of the search space.
- *Avoid overwhelming solvers:* each solver maintains a database containing both locally-learned lemmas and lemmas imported from the broker. Core solver performance degrades as the size of the database grows. Sharing too many lemmas can thus be detrimental to overall system performance.
- *Manage communication overhead:* we do not want to overload the communication network with too much data, as this also slows down the system.

Our proposed architecture supports a wide variety of strategy options. We mention two general strategies here, and then discuss specific parameter settings used in our implementation in Section 4.5. The first strategy is *delayed sharing*, which avoids sharing a large set of lemmas that all solvers discover locally. The second strategy is a novel approach to diversity that we call *guided randomization*.

4.4.1 Delayed Sharing

In initial experiments with an early prototype, we observed that for some large problems, workers initially export a large number of lemmas and delay calling the *importLemmas* procedure. Later, when they do try to import the lemmas, the system stalls due to the large amount of communication traffic. Telemetry revealed that this was caused by the initial preprocessing and theory reasoning performed by the solvers.

Before entering the CDCL loop proper, SMT solvers perform formula simplification, conversion to clausal form, and some eager theory reasoning. It is possible for solvers to produce many lemmas during this phase; if each worker is an instance of the same SMT solver, such lemmas are likely to be learned by all solvers working on the problem.To address this issue, we added a delayed sharing mechanism, which ensures that only lemmas learned *after* the preprocessing phase are exported. Enabling this mechanism boosts performance on all of our benchmarks.

4.4.2 Guided Randomization

Baseline mechanisms for diversifying solver behavior include selecting different random seeds and modifying solver configurations to ensure that different instances use different search parameters. However, these basic mechanisms have diminishing benefit as we increase portfolio size, as we show in Section 4.6.1. Using the telemetry collected by the broker, we can observe the number of uniquely learned lemmas (i.e., those learned by a single worker). This metric is a reasonable proxy for system diversity, and indeed, in early experiments, we observed that this number plateaus as we scale the number of workers.

We address this problem by dividing the pool of workers into two clusters, a standard cluster and a *noisy* cluster. Each cluster uses different levels of randomness and different scoring and filtering heuristics. Scoring and filtering can also treat lemmas local to the cluster differently than clauses from other clusters. The noisy cluster uses a high degree of randomness. Intuitively, we expect that solvers in this cluster will learn mostly useless clauses, because they are using heuristics that are far away from the default configurations

which have been tuned to be effective. They are also likely to end up exploring parts of the search space that low-randomness solvers ignore. But once in a while, noisy solvers may get lucky and learn clauses that can be useful to solvers in the other cluster.

To maintain diversity in the noisy cluster, we keep the clause databases for solvers in the cluster somewhat isolated. We do this by configuring solvers in noisy clusters to ignore each other and only import lemmas that the central manager determines are highly likely to be useful, (*e.g.,* unit clauses). We discuss a concrete instantiation of this strategy in the next section.

4.5 Implementation

CVC5-D is a distributed SMT solver that implements our proposed architecture and strategies. For the worker instances, we use a version of cvc5 with the main loop modified to support importing and exporting clauses, as discussed in Section 4.3.1. Workers run in separate processes, and each worker process has a separate *wrapper* thread that manages the control plane interface and networking details.

The central manager is written in Python. Communication between broker and workers is implemented with gRPC [48]. We chose gRPC instead of lower-level mechanisms like sockets, because gRPC's high-level API provides better monitoring capabilities and has sufficient performance for (at least) 64 solvers. gRPC also allows us to abstract the parallel and distributed aspects of the system. Thus, CVC5-D can be deployed either on a single multicore machine or on a cluster of machines in the cloud.

To export lemmas, we serialize them as strings in the SMT-LIB format [11]. Correspondingly, lemma import requires parsing SMT-LIB strings. This adds some overhead but provides a significant interoperability advantage, as all SMT solvers can parse and print terms in SMT-LIB format. More compact formats could be used at the cost of increased implementation effort and reduced interoperability. For example, SMTS uses a dedicated binary format, but this limits the choice of solvers to those that support this format. Choosing SMT-LIB reduces the cost of adding solvers beyond cvc5 to CVC5-D.

As explained previously, CVC5-D implements comprehensive telemetry for both the control and data planes. We found this real-time information about the solving process at both the local and global levels to be crucial when debugging the system, evaluating different portfolio configurations, and evaluating lemma scoring and filtering strategies. The implementation is heavily parameterized, so that whenever possible, users can choose

configuration options at runtime, rather than having to change hard-coded configuration settings.

Local Filtering Several considerations must be taken into account at the worker level. SMT solvers can dynamically create new atoms and new variables during search. This poses a soundness problem in a distributed setting as one must ensure that new variables created by a solver instance are interpreted consistently by other instances. We currently avoid this issue at the export stage by filtering out lemmas that contain variables not present in the original formula. New theory atoms are fine as long as they do not introduce new variables. More sophisticated approaches are possible, but require a mechanism for exporting the definitions of new variables in a canonical way. Implementing such a mechanism requires extending the baseline SMT solver in a non-trivial way, and we leave it for future work.

As mentioned, our primary goal when filtering is to only export useful lemmas. As in prior work, we use the number of literals in the lemma as our main export filter.

Importing lemmas has a cost. The central manager aims to limit redundancy by only sending a given lemma once to each worker. It is still possible for a worker to produce a lemma internally before learning that another worker has produced the same lemma. Thus, we check in the import procedure whether an imported lemma has already been discovered locally. If so, we drop it. This can be implemented efficiently using mechanisms such as hashing and Bloom filters.

Sending Lemmas from the Manager Our broker uses two datasets to determine when to send lemmas. The first is the wall clock time elapsed since the last lemma transmission. The other is the number of unsent lemmas for a particular worker in the *lemmaSolverMap* map. Function *shouldSend* returns true if the elapsed time is greater than a parameter *delay* or if the number of unsent lemmas is larger than a threshold *maxQueueSize*. By setting these two parameters, the broker can implement different communication policies. It can send lemmas in size-driven batches (like SMTS [83]), in time-driven epochs (like Mallob [107]), or both. We found empirically that so far, the best results come from sharing lemmas individually as soon as they are received. If we encounter network bandwidth limitations at some point, we expect that time-driven epochs will provide the best efficiency.

Monitoring CVC5-D uses telemetry from the workers to monitor the number of lemmas imported and exported by each worker. Information from solver wrappers is used to monitor message latency and manager/solver roundtrip times. The *lemmaSolverMap* map also

Benchmarks		CVC5-D 64x CS-GR		SMTS baseline		SMTS 64x CS		CVC5-P 64x	
Category	Count	Solved	PAR-2	Solved	PAR-2	Solved	PAR-2	Solved	PAR-2
QF_LRA	139	120	60 (↓**61%**)	117	69	**127**	**41** (↓41%)	99	130 (↓16%)
QF_IDL	48	**21**	**70** (↓**39%**)	8	99	15	82 (↓17%)	5	107 (↓6%)
QF_LIA	16	9	20 (↓**47%**)	11	13	**14**	**11** (↓15%)	1	36 (↓5%)
QF_UF	7	**7**	**2** (↓**86%**)	6	5	6	3 (↓40%)	4	9 (↓36%)
QF_RDL	4	**2**	**6** (↓**40%**)	0	10	0	10 (0%)	0	10 (0%)
SAT	115	86	82 (↓**52%**)	83	87	**99**	**44** (↓49%)	59	151 (↓12%)
UNSAT	85	**73**	**43** (↓**65%**)	59	75	63	63 (↓16%)	50	106 (↓15%)
UNKNOWN	14	0	34 (0%)	0	34	0	34 (0%)	0	34 (0%)
ALL	214	159	159 (↓**52%**)	142	196	**162**	**141** (↓28%)	109	291 (↓12%)

Table 4.1: Results comparing the best config of CVC5-D with different distributed solvers. PAR-2 scores in thousands. Numbers in brackets denote how much different distributed solvers improve over their base solver.

Benchmarks		CVC5-D baseline		CVC5-D 64x CS		CVC5-D 64x CS-GR	
Category	Count	Solved	PAR-2	Solved	PAR-2	Solved	PAR-2
QF_LRA	139	90	154	121	61 (↓60%)	120	60 (↓**61%**)
QF_IDL	48	1	114	20	72 (↓37%)	**21**	**70** (↓**39%**)
QF_LIA	16	0	38	8	22 (↓42%)	9	20 (↓**47%**)
QF_UF	7	2	14	3	11 (↓21%)	**7**	**2** (↓**86%**)
QF_RDL	4	0	10	2	6 (↓40%)	**2**	**6** (↓**40%**)
SAT	115	52	172	86	83 (↓52%)	86	82 (↓**52%**)
UNSAT	85	41	124	68	55 (↓56%)	**73**	**43** (↓**65%**)
UNKNOWN	14	0	34	0	34 (0%)	0	34 (0%)
ALL	214	93	330	154	171 (↓48%)	159	159 (↓**52%**)

Table 4.2: Results comparing different configs of CVC5-D. PAR-2 scores in thousands

tracks how many solvers independently learned each lemma, *e.g.,* the number of lemmas learned by exactly one solver, two solvers, and so forth. This helps dynamically measure diversity in the system, including the amount of redundant work being performed by different solvers. The broker also maintains its own counts of the number of exported and imported lemmas for each worker. Mismatches between the numbers stored in the broker and the numbers reported by the workers mean that the system is overloaded (thus messages are late or dropped) or that there is a bug. During the development of CVC5-D, the monitor helped detect multiple bugs and helped inform the design of our lemma-sharing heuristics.

4.6 Evaluation

We measure CVC5-D performance on the set of benchmarks used in [136], which consists of 214 challenging benchmarks taken from the Cloud track of SMTCOMP22 and the set of QF_LRA and QF_UF benchmarks in SMT-LIB. The benchmarks come from five SMT-LIB logics: QF_LRA (139), QF_IDL(48), QF_LIA (16), QF_UF (7), and QF_RDL (4).

We use a competition build of cvc5 with the optional CLN and GLPK options enabled. For cvc5 portfolios, we use several different sets of options in order to improve diversity.

In all experiments, we set the timeout for solving each query to be 1200 seconds, the same timeout used in SMT-COMP. Experiments were performed on Amazon EC2 c6a.48xlarge instances, with 96 physical cores and 384 GB of RAM.

Our main metric used for comparison is the *PAR-2 score* used in [136] and the annual SAT competition. PAR-2 is the sum of run times for all instances, but where unsolved instances receive a score of twice the timeout value ($1200 \times 2 = 2400$). This provides a single metric that takes into account both runtime and number of benchmarks solved. The lower the PAR-2 score, the better. We also use *cactus plots* to show the number of solved instances (y-axis) within a limit of s seconds per instance (x-axis). We are primarily interested in the effectiveness of different parallelization strategies and implementations.

4.6.1 Scalability and Effectiveness of Guided Randomization

We first report on scalability experiments of CVC5-D, both with and without sharing. We also show the effect of adding guided randomization. When using guided randomization, we divide the portfolio into two clusters: a *standard* cluster, which uses default cvc5 randomness settings, and a *noisy* cluster, which assigns the cvc5 `rnd_freq` option to 75%. This option controls how often the SAT decision tries to pick a random variable instead of a heuristically-driven choice. We assign 25% of the workers to the noisy cluster and 75% to the standard cluster. Solvers in the standard cluster import and export clauses of length ≤ 8. In the noisy cluster, clauses of length ≤ 4 are exported, but only unit clauses are imported.

To distinguish the different configurations of CVC5-D, we use *CS* for configurations with clause sharing and *CS-GR* for configurations with clause sharing and guided randomization. Fig. 4.2 shows how different configurations of CVC5-D scale with the number of workers. The figure includes results for baseline cvc5, portfolio sizes of 4, 16, and 64, both with and without sharing, as well as an additional run with 64 workers with guided randomization.

Specific numbers for three of the configurations (baseline, 64x CS, and 64x CS-GR) can be found in Table 4.1.

We observe that CVC5-D scales nicely when going from 1 to 64 solvers. In addition, clause sharing improves performance for all portfolio sizes greater than four, and guided randomization provides an additional boost. A comparison of the 64x CS configuration with and without guided randomization is shown in Fig. 4.3. We can see that CS-GR is especially effective on satisfiable (SAT) instances. CS-GR also improves the performance by more than 2x for many problems (dots to the left of the top "2x" line). As a whole, among all instances solved by both CS and CS-GR, there are 24 instances where CS-GR is more than 2x faster than CS, and only 5 instances where CS-GR is 2x slower. CS-GR solves 5 more problems, and improves PAR-2 score by 12k (7%) over CS.

4.6.2 Comparison with State-Of-The-Art Tools

We next compare CVC5-D with SMTS [84], the strongest solver in quantifier-free divisions of SMTCOMP22's cloud track,[1] and CVC5-P, the partitioning solver from [136].

Comparison to smts It is important to note that on this benchmark set, OPENSMT2, the baseline solver for SMTS, is stronger than cvc5.[2] However, the best configuration of CVC5-D (64 CS-GR) improves this situation significantly. Table 4.1 shows that overall, in terms of benchmarks solved, the best configuration of CVC5-D (64 CS-GR) is roughly comparable to the best configuration of SMTS, despite the large difference in their base solvers. Compared to the baseline, the best configuration of CVC5-D improves the overall PAR-2 score by 52% (for SMTS, this number is only 28%) and solves 66 more problems (compared to 20 more problems solved by SMTS). Moreover, for the 48 QF_IDL benchmarks and for the UNSAT benchmarks as a whole, cvc5 goes from performing worse than SMTS when comparing baselines to performing better when comparing the best version of each. This suggests that at least part of the reason for the improvement is explained by our lemma-sharing implementation being more effective.

Comparison to partitioning cvc5 CVC5-P, the state-of-the-art parallel/distributed implementation of cvc5, uses a combination of portfolios and partitioning strategies. We

[1] SMT-COMP 2023's cloud track omitted all quantifer-free divisions.

[2] One reason for this is that the benchmarks we are using, from [136], were selected specifically because they are challenging for cvc5.

implemented and ran the hybrid multijob approach of [136] and compared it with CVC5-D. Fig. 4.7 and Table 4.1 show that CVC5-D is significantly more effective at utilizing 64 copies of cvc5, resulting in a 52% improvement in PAR-2 score (vs 12% improvement by CVC5-P), and in 50 more problems being solved (159 vs 109).

4.6.3 Comparison to a Legacy Version of z3

Z3 was the first SMT solver to implement a portfolio approach with clause sharing. However, this functionality is no longer supported in modern versions of Z3, and the latest release that we could find with this functionality is version 2.15 (Windows-only, from 2009). We include the comparison here for completeness, but with two caveats: first, Z3 2.15 runs on a different operating system than our other solvers (we used instances with the same ratio of workers to processor cores), and it crashes on any configuration with more than eight solvers. We note that Z3 2.15 fails (parsing or execution) on 85 problems in our modern set of 214 benchmarks. For this reason, we only compare Z3 2.15 with CVC5-D using eight workers on the remaining 129 SMT benchmarks. We do not enable guided randomization here because CVC5-D does not saturate diversity at 8 solvers. Fig. 4.5a show that CVC5-D scales to eight solvers more effectively than Z3 2.15 and as a whole achieves better PAR-2 and solves more problems than Z3 2.15. Note that Z3 performs *worse* when enabling clause sharing, indicating the instability of the 2.15 implementation on modern benchmarks. A fair comparison could only be achieved if the sharing functionality were restored in a modern version of Z3.

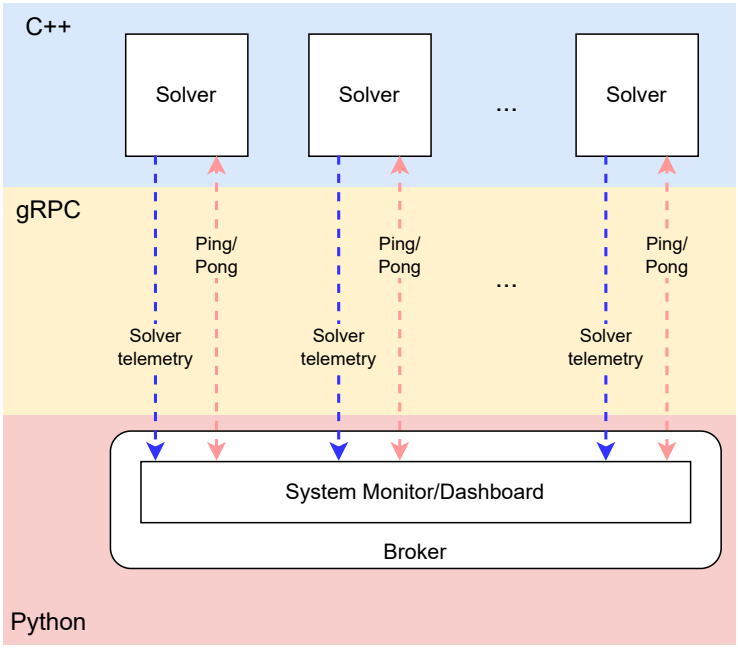
4.7 Conclusion

CVC5-D is a promising advancement in the realm of parallel, portfolio-based SMT solving. Leveraging a hub-and-spoke architecture with a tight CDCL(T) integration, lemma sharing, and guided randomization, CVC5-D demonstrates significant improvements in scalability, outperforming not just sequential cvc5, but also pure portfolio (with sharing), and CVC5-P (portfolio with partitioning). In addition, CVC5-D demonstrates more improvement from clause sharing than SMTS and an early version of Z3 and has performance that is overall comparable with and complementary to the state of the art.

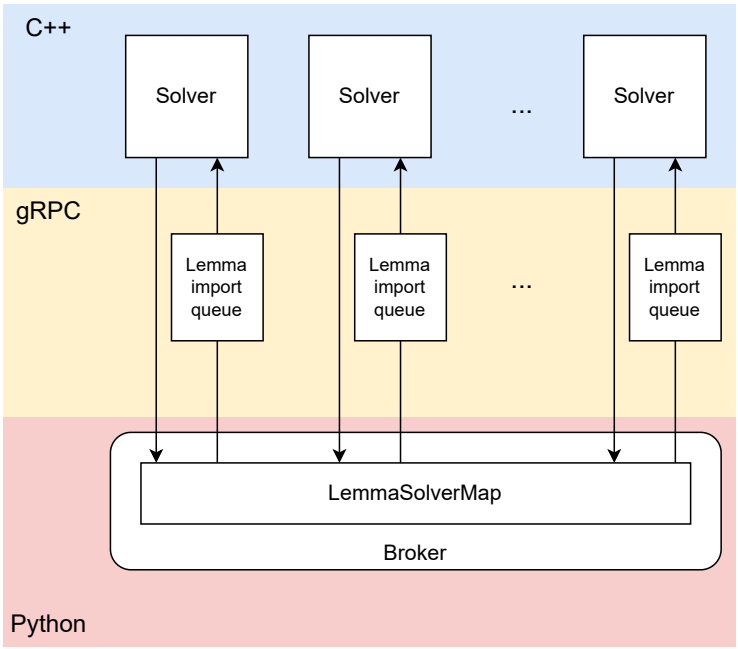
While CVC5-D demonstrates solid progress in distributed SMT solving, many opportunities for future work remain. These include deeper integration with the underlying SAT solver, handling internally-introduced variables, exploring additional sources of diversity, and combining our approach with partitioning-based parallelism.

This chapter is adapted from the following work:

- Clark Barrett, Pei-Wei Chen, Byron Cook, Bruno Dutertre, Robert Jones, Nham Le, Andrew Reynolds, Kunal Sheth, Chriss Stevens, and Mike Whalen. CVC5-D: New Strategies for Portfolio-Based SMT Solving. Accepted at The Twenty-fourth Conference on Formal Methods in Computer-Aided Design. 2024, Prague, Czech Republic.



(a) Control Plane



(b) Data Plane

Figure 4.1: Architecture of CVC5-D

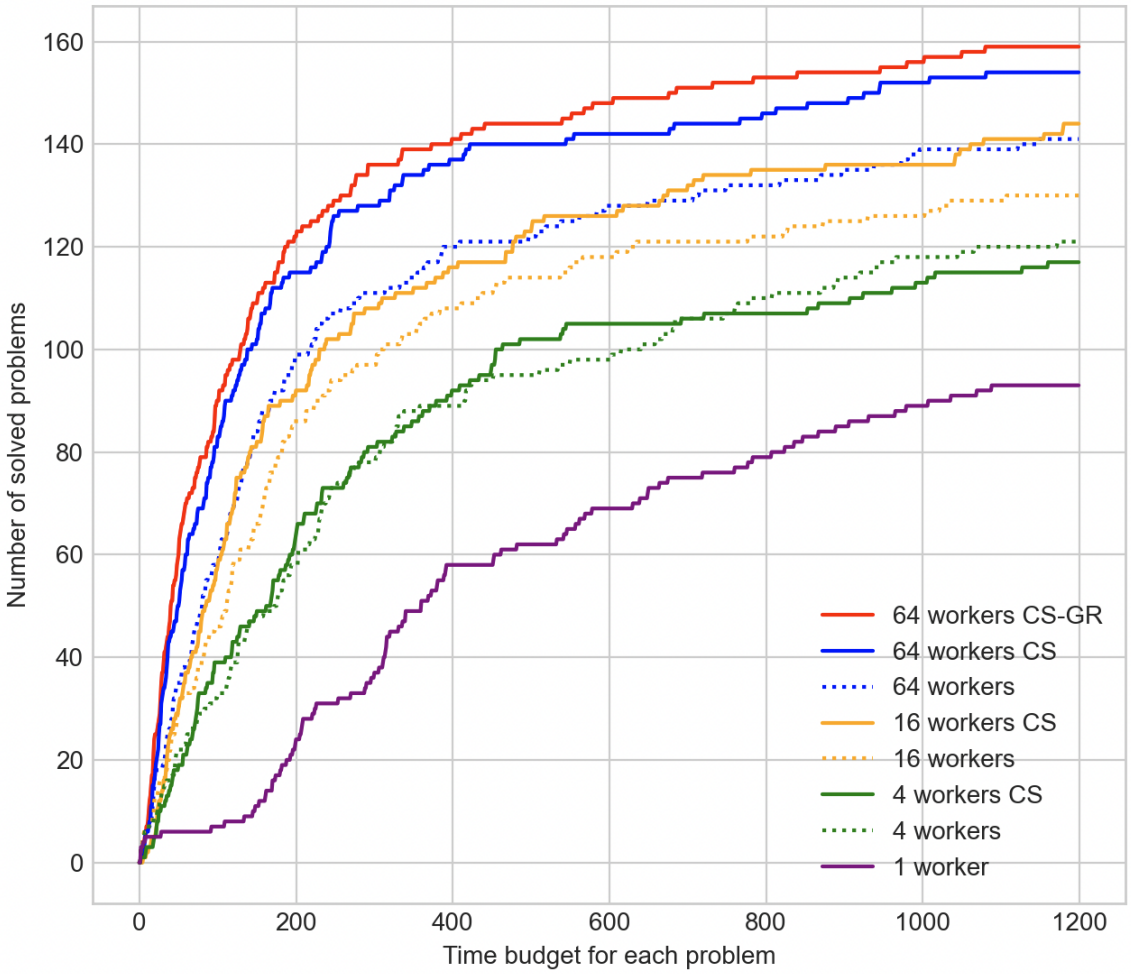


Figure 4.2: Scalability of CVC5-D.

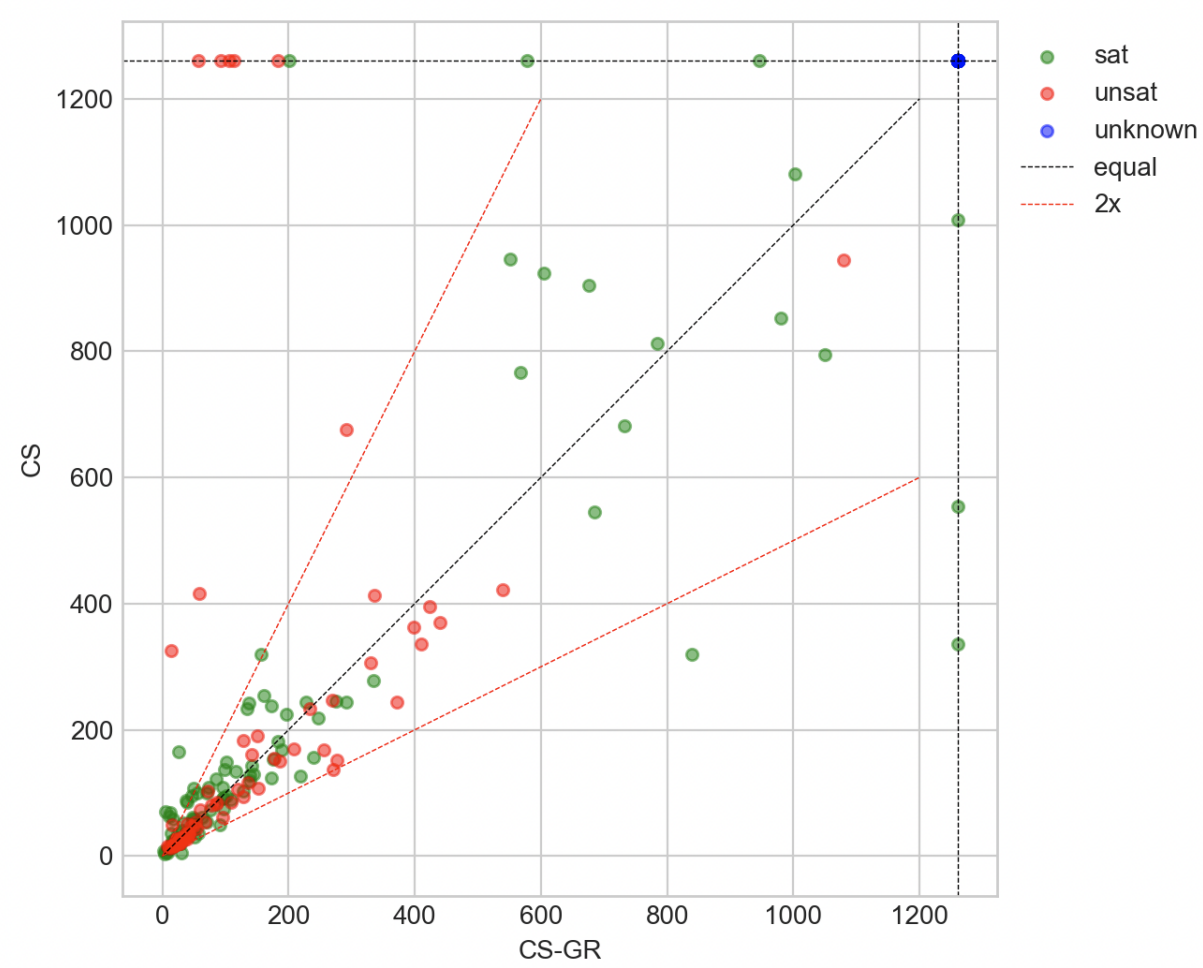
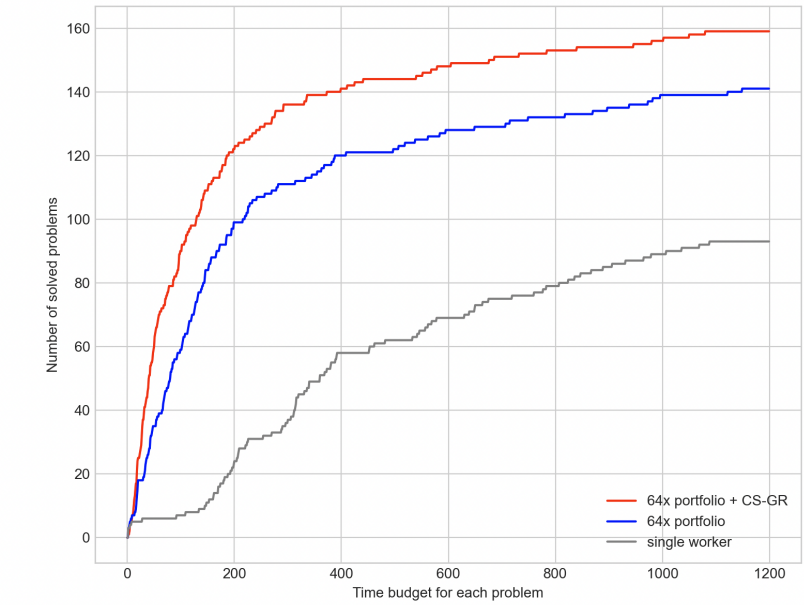
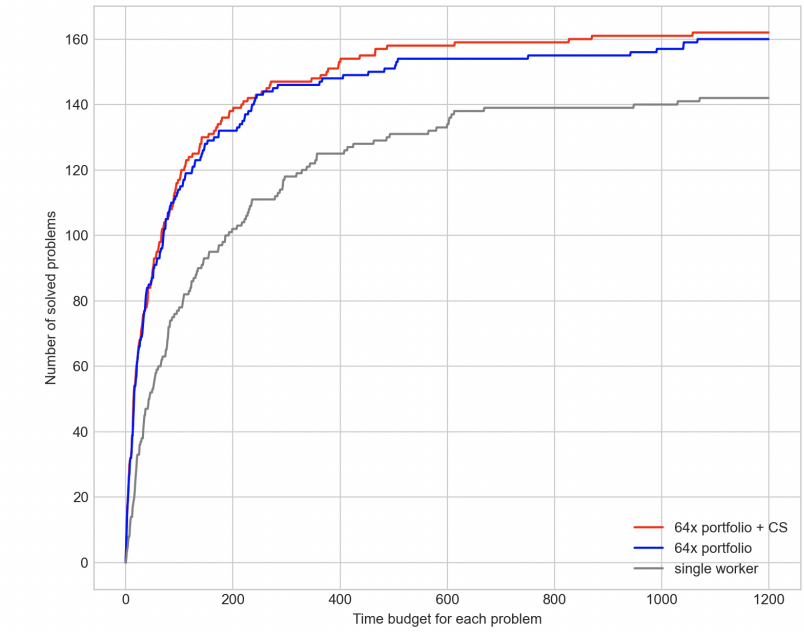


Figure 4.3: Guided Randomization (CS-GR) vs naive Clause Sharing (CS). Dots on the upper and right-most edges are problems that time out with CS and CS-GD, respectively.

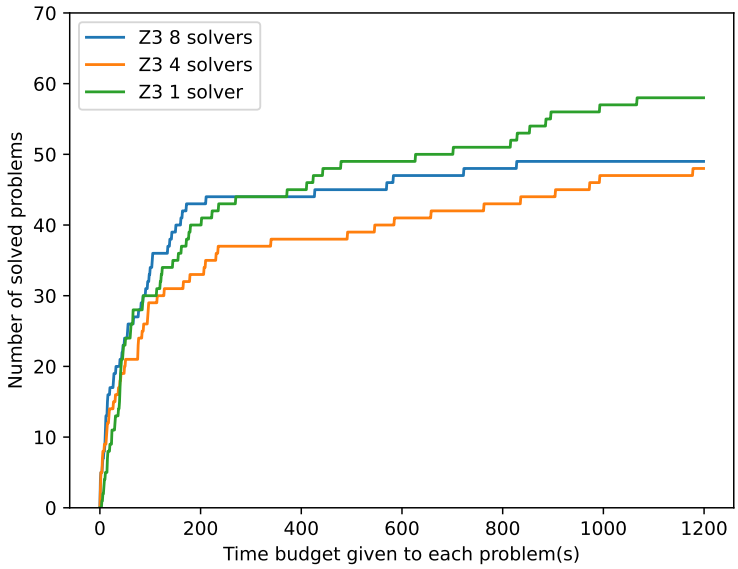


(a) CVC5-D

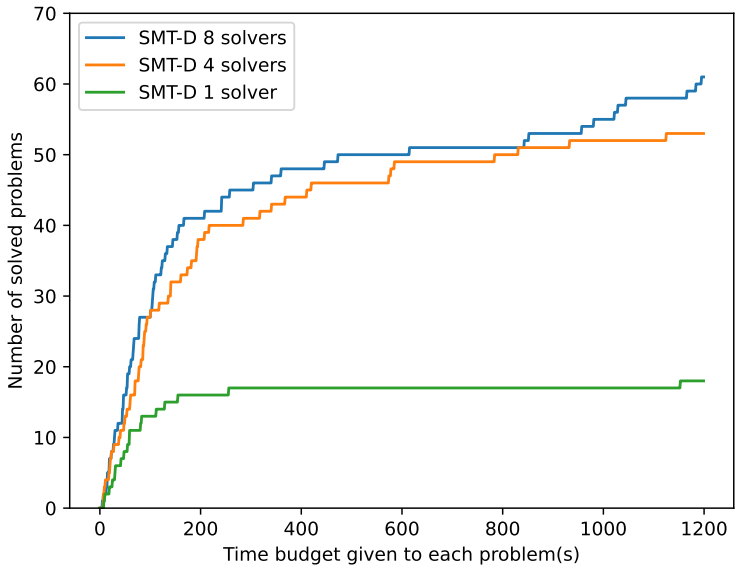


(b) SMTS

Figure 4.4: Comparing CVC5-D's and SMTS' improvement over a single base solver.



(a) Z3



(b) CVC5-D

Figure 4.5: Comparison between CVC5-D and Z3 on 129 benchmarks).

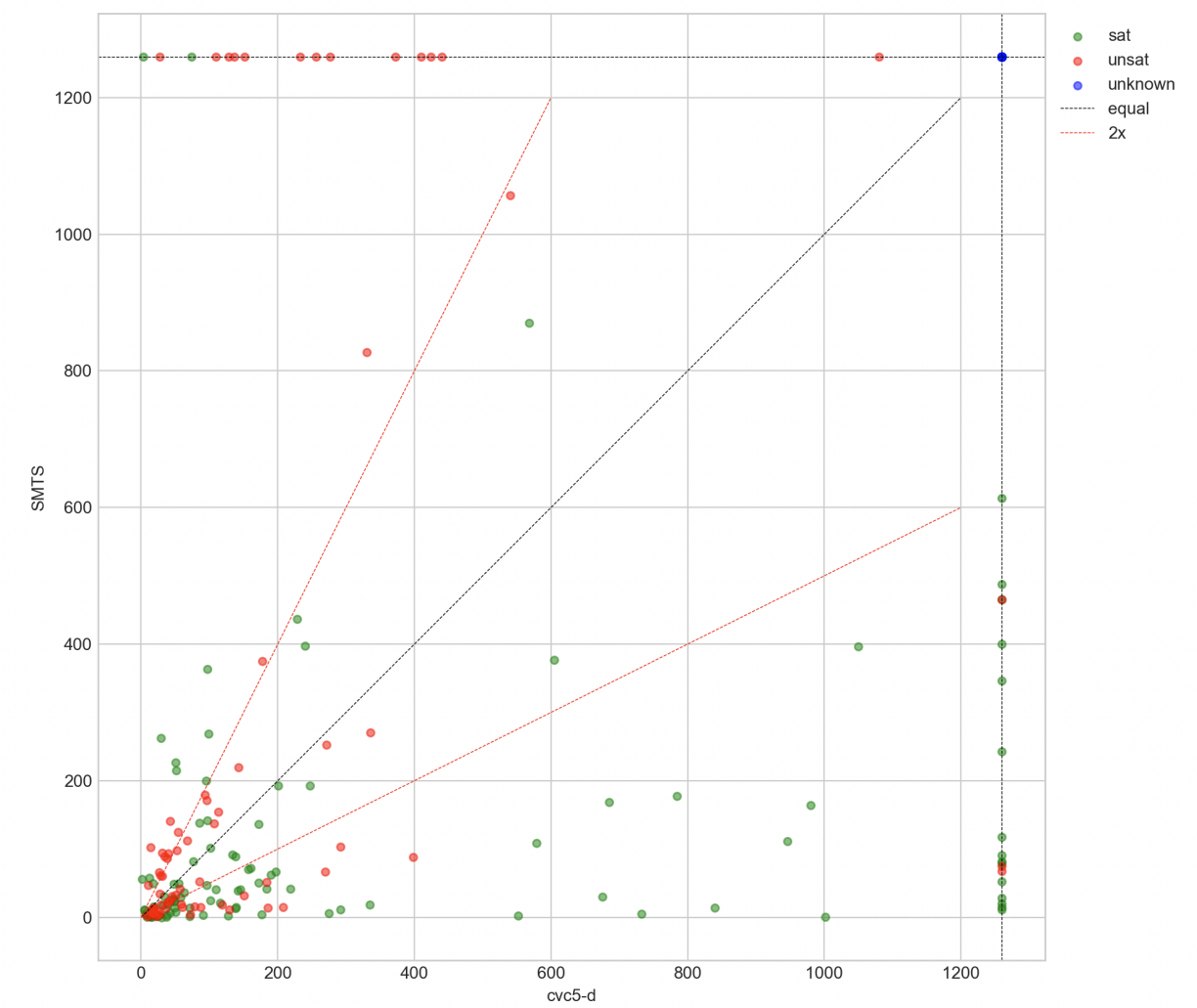


Figure 4.6: CVC5-D vs SMTS. Dots on the upper and right-most edges are problems that time out for SMTS and CVC5-D, respectively.

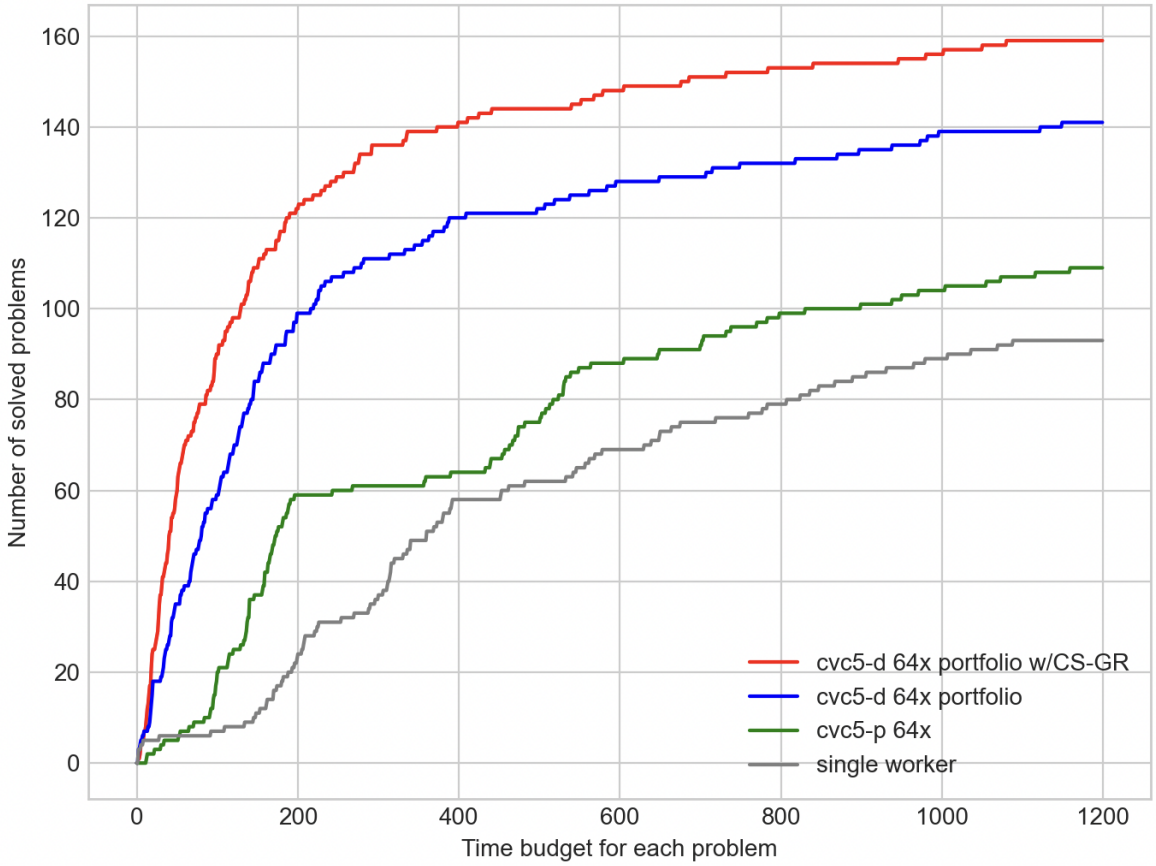


Figure 4.7: CVC5-D and CVC5-P, 64 workers vs 1 worker.

Chapter 5

Verifying the robustness properties of saliency maps

Saliency maps are one of the most popular tools to interpret the operation of a neural network: they compute input features deemed relevant to the final prediction, which are often subsets of pixels that are easily understandable by a human being. However, it is known that relying solely on human assessment to judge a saliency map method can be misleading.

In this chapter, we propose a novel neural network verification specification called *saliency-robustness*, which aims to use formal methods to prove a relationship between Vanilla Gradient (VG) – a simple yet surprisingly effective saliency map method – and the network's prediction: given a network, if an input x emits a certain VG saliency map, it is mathematically proven (or disproven) that the network must classify x in a certain way. We then introduce a novel method that combines both Marabou and Crown– two state-of-the-art neural network verifiers, to solve the proposed specification. Experiments on our synthetic dataset and MNIST show that Vanilla Gradient is surprisingly effective as a certification for the predicted output.

5.1 Introduction

As deep neural networks (DNNs) continue to advance in complexity and impact, the demand for explanation methods and tools to interpret key aspects of these models also grows. The ability to explain how a model operates can be crucial in meeting regulatory

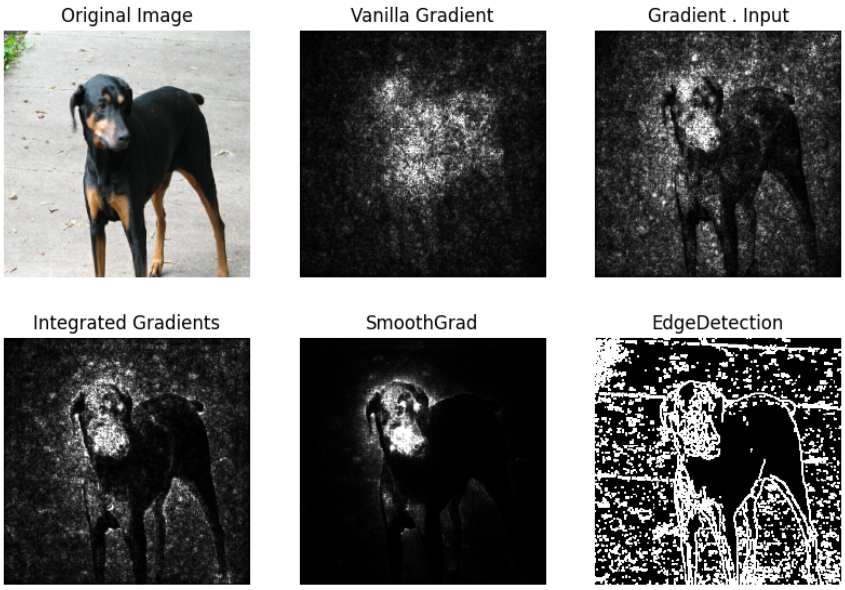


Figure 5.1: Different saliency map methods of a dog, together with the result of an edge detection algorithm [22] that does not take the model into account at all.

requirements [47] and assisting developers in debugging the model [73]. Among the various explanation methods available, one category that stands out is saliency maps [117, 109, 71], primarily due to their interpretability. Saliency maps identify input features that are considered relevant to the final prediction, often highlighting specific pixels that can be easily understood by humans. Fig. 5.1 visualizes an image of a dog and how some different saliency map methods highlight pixels that are deemed important. However, the abundance of different saliency map methods raises a methodological question for practitioners: how does one choose between these numerous options?

Thus far, the evaluation of most saliency map methods has heavily relied on subjective human judgment. The assessment typically follows the approach of "the saliency map is considered good if it appears visually appealing to me". While it is necessary to discard obviously inadequate methods through a sanity check [2], users of saliency maps are left to select the appropriate method based on their own visual assessment. As pointed out in [2], while we want an explanation to take into account *both* the input and the network, human judgment tends to be biased toward the input. This presents a problem as humans may inadvertently focus on explaining the input itself rather than understanding the relationship between the input and the model. The issue is highlighted in Fig. 5.1, where the Canny

edge detector algorithm [22] (which does not consider the DNN at all and should not be used to explain any DNN) produces a visually convincing map that bears a resemblance to those generated by some other saliency methods.

In this chapter, we propose to use *formal methods* to mathematically prove or disprove a relationship between a saliency map and the prediction of the network. As the first work in this direction, we apply our method with Vanilla Gradient (VG) [5] – an elegant yet surprisingly effective formulation of saliency maps: it passes all of the sanity checks proposed by [2] while several more modern methods [122, 124, 115] do not.

Our key insight is our novel concept of *saliency-robustness*, which states that if the Vanilla Gradient saliency map $E_{\mathcal{N}}$ for the network $\mathcal{N}$ is reliable, then two images generating similar saliency maps should be classified in the same manner by $\mathcal{N}$. This property is essential because it ensures that $E_{\mathcal{N}}$ can genuinely explain why an image belongs to the label "dog" rather than the label "cat". Conversely, if this property does not hold, then $E_{\mathcal{N}}$ may not provide accurate explanations. To the best of our knowledge, this is the first time in which a *mathematically proven* relationship between a saliency map function and a prediction is attempted.

To solve our proposed saliency-robustness property, we make another insight: computing the saliency map of a ReLU-activated neural network can be done by solving a system of linear constraints. Thus, the whole property can be encoded as a set of linear constraints and solved using an off-the-shelf SMT solver. While this is acceptable as a proof of concept, it is widely known that off-the-shelf SMT solvers do not scale to solving neural networks of interesting sizes [69]. To overcome that challenge, we propose a novel method to solve the saliency-robustness property more effectively, by combining state-of-the-art techniques in neural network verification. To sum up, we make the following contributions:

- We propose a novel safety property for a neural network and its Vanilla Gradient saliency map, called *saliency-robustness*.
- We show that the proposed property can be verified by solving a constraint satisfiability problem over linear real arithmetic (LRA).
- We propose a novel method to solve the saliency-robustness problem more effectively, by combining two state-of-the-art techniques in neural network verification, namely constraint-based solving [70] and Jacobian bounding [147, 133, 114].
- We conduct experiments on our synthetic benchmarks and dataset and a neural network from VNNCOMP23 [91], the annual neural network verification competition.

	Z3	Ours
$\delta = 0.5$	3.22s UNSAT	0.8s UNSAT
$\delta = 0.75$	2.8s UNSAT	1.225s UNSAT
$\delta = 1$	2.9s UNSAT	1.258s UNSAT
$\delta = 1.25$	3.36s SAT	2.212s SAT
$\delta = 1.5$	2.78s SAT	0.8s SAT

Table 5.1: Verifying saliency-robustness for BanditNet using Z3 and Marabou +Crown

> We find that Vanilla Gradient, despite being the earliest form of the saliency maps, is a surprisingly good explanation for the tested network.

The rest of the chapter is structured as follows: Section 5.2 provides a concrete example as well as describes our synthetic dataset, Section 5.3 goes into details our proposed saliency-robustness property and how to solve it, Section 5.4 presents our experiments and results, and finally Section 5.5 summarizes our contributions, outlines the current limitations of the method, and discusses open problems for future work.

5.2 A motivating example

In this section, we provide a concrete example to illustrate our idea. We consider a multi-arm bandit machine with 5 arms, each capable of generating a specific reward by manipulating its complete state. However, unlike digital arms, these arms are analog and can be pulled at varying levels of intensity, ranging from 0% to 100%. For instance, if an arm has a reward value of 300, pulling it at 10% intensity will result in a reward of 30. The rewards for the five arms are as follows: 100, 100, 300, 100, 300.

To obtain the total reward, the player must pull each arm to an arbitrary level. We can represent the machine configuration with a five-element vector. For instance, if the machine has only the first two arms pulled to 50% level, this configuration can be represented by the vector $[0.5, 0.5, 0., 0., 0.]$. The total reward is simply the sum of rewards obtained from each arm. We consider the total reward greater than 300 to be a high reward, and anything less to be a low reward. However, this information is not revealed to the players. Suppose a player records several configurations of complete arm states, such as $[1.0, 0., 0., 1., 0.]$. The player has a dataset of 20 configurations. They can then train a simple FCN model to predict the corresponding reward outcome. By analyzing the saliency map, the player

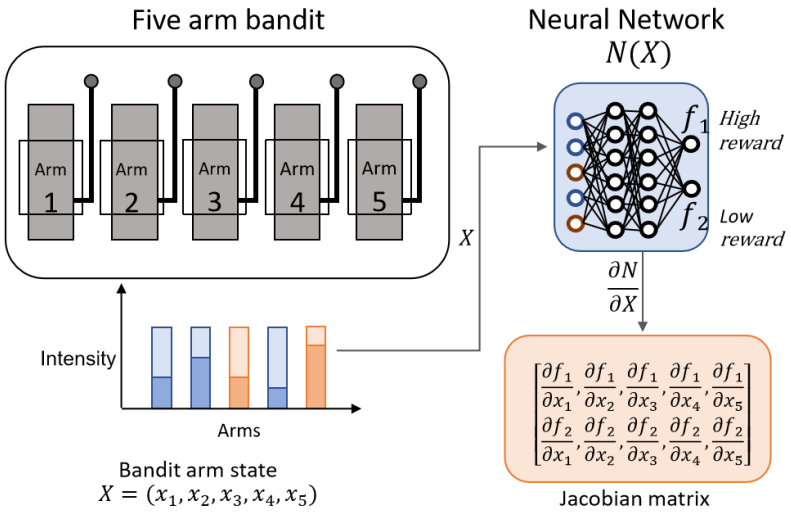


Figure 5.2: Five arm bandit.

realizes that the third and fifth arms have the highest absolute value of gradients. They can take advantage of this information to improve their strategy. But they have a burning question: is the saliency map truly an explanation for the predicted outcome or a mere correlation?

The user looks at the saliency map of the input $[1, 1, 0, 1, 1]$ with respect to the "high reward" label, which is $M = [0.03, 0.23, 2.97, 0.05, 2.5]$. If the saliency map is an explanation for the prediction, then for all input in $[0, 1]^5$, a saliency map similar to M must imply that the output is classified as "high reward", the user figures. They look at M and see that the gap between the high and low values is about 2.5, thus they expect that for all saliency maps that have L_∞ distance to M of less than 1.25 (i.e saliency maps with the same two arms being highlighted), they should all guarantee the prediction of "high reward". Using our method, they verify that it is indeed the case, as shown by Table 5.1.

5.3 Methodology

In this section, we introduce our new verification problem, called *the saliency-robustness problem*, how solving it can be seen as solving a constraint satisfiability problem over LRA, and how to effectively solve it by combining state-of-the-art techniques in neural network verification.

5.3.1 The saliency-robustness problem

Given the aim of a saliency map E, we ask the question: if $E(x)$ and $E(x')$ are "similar" (according to some metrics or human judgment), must $\mathcal{O}(x) = \mathcal{O}(x')$? If that's not the case, then E is hardly a good explanation: if the same set of pixels are important for both recognizing digit 0 and 1, then that set of features cannot be used to explain why an image is of the label 0 but not 1.

We formalize this question by the following verification problem

$$\forall x, x' \in \mathbb{R}^{d_0} \cdot \mathcal{O}(x) = \ell \wedge E(x') \approx E(x) \implies \mathcal{O}(x') = \ell \tag{5.1}$$

in which $E(x) \approx E(x')$ indicate that they are similar. There are many different ways of defining similarity, and we leave exploring different formulations for future work. In this chapter, we use the same notion of similarity that is commonly used in robustness verification [3]: two saliency maps are similar if they are close in the L_∞ norm.

The quantifier in Eq. (5.1) reflects the ideal scenario in which the property can be verified in the whole input domain. In practice, this is rarely the case given the scalability of existing tools. Thus, we aim to solve a ϵ-relaxed problem and aim to push the parameter ϵ higher in future work. Concretely, we verify inputs in the *epsilon* vicinity of datapoints (similar to the robustness problem): given a target input $\hat{x}$, we check the following *saliency-robustness* property

$$\forall x' \in B(x, \epsilon) \cdot \mathcal{O}(\hat{x}) = \ell \wedge ||E(x') - E(\hat{x})||_\infty \leq \delta \tag{5.2}$$

$$\implies \mathcal{O}(x') = \ell \tag{5.3}$$

If Eq. (5.2) holds, we say that E is (ϵ, δ)-robust at $\hat{x}$. In our motivating example, the user wants to check queries ranging from $(1, 0.5)$- to $(1, 1.5)$-robustness of at $\hat{x} = [1, 1, 0, 1, 1]$.

Note that per our definition in Chapter 2, E is a 2D matrix computing the gradient of each input with respect to each label. In many saliency map methods [109, 118, 71], it is common to focus on only the gradient with respect to the label with the highest score ($\mathcal{O}(x)$). From this point on, unless specified otherwise, we consider E as a gradient *vector* with respect to the predicted label.

5.3.2 The saliency-robustness as a constraint satisfiability problem

We show that for any neural network consisting of only linear layers and piecewise-linear activation functions, Eq. (5.2) can be encoded into a satisfiability problem over linear real arithmetic (LRA). First, given a target input $\hat{x}$ and a neural network $\mathcal{N}$ that predicts $\mathcal{O}_{\mathcal{N}}(\hat{x}) = \ell$, we define the following first-order logic formula

$$f = \phi_F \wedge \phi_G \wedge \phi_{\approx} \wedge \phi_P$$

in which

ϕ_F = Constraints for forward computation (encoding $\mathcal{O}_{\mathcal{N}}(x)$)

ϕ_G = Constraints for computing gradient (encoding $E_{\mathcal{N}}(x)$)

$\phi_{\approx}$ = Constraints for gradient similarity

ϕ_P = Constraints for ϵ-robustness

The forward computation can be encoded using the same encoding used by MARABOU [70]: for a linear layer $z^i = \boldsymbol{W}^i h^{i-1} + \boldsymbol{b}^i$, we have the constraint $z^i[j] = \sum_{k=1}^{|h_{i-1}|} \boldsymbol{W}^i[j][k] + \boldsymbol{b}^i[j]$ for each entry in the resulting layer z^i; and for a ReLU activation layer $h^i = ReLU(z^i)$, each entry in the result vector can be encoded using two implications: $z^i[j] > 0 \implies h^i[j] = z^i[j]$ and $z^i[j] \leq 0 \implies h^i[j] = 0$.

The backward computation can be encoded recursively as follows. For each layer h^i and z^i, we denote ∂h^i and ∂z^i their gradient vectors. At the last layer z^L, we set $\partial z^L[\ell] = 1$, and $\partial z^L[j \neq \ell] = 0$.

For the linear layer $z^i = \boldsymbol{W}^i h^{i-1} + \boldsymbol{b}^i$, the j^{th} entry in the gradient of ∂h^{i-1} is computed by

$$\partial h^{i-1}[j] = \sum_{k=1}^{|z^i|} \boldsymbol{W}^i[k][j] \partial z^i[k] \tag{5.4}$$

The backward computation for the convolutional layer (which is a specialized version of the linear layer) can be encoded in a similar manner.

For the ReLU layer $h^i = ReLU(z^i)$, the j^{th} entry in the gradient of ∂z^i is encoded by two implications

$$z^i[j] > 0 \implies \partial z^i[j] = \partial h^i[j] \tag{5.5}$$
$$z^i[j] \leq 0 \implies \partial z^i[j] = 0 \tag{5.6}$$

The gradient similarity is encoded as bounds on each entry in the vector ∂h^0. Given the precomputed $E(\hat{x})$ (which can be computed using any off-the-shelf auto-gradient tools like Pytorch or Tensorflow), we set

$$\forall j \in [1, d_0] \cdot E(\hat{x})[j] - \delta \leq \partial h^0[j] \leq E(\hat{x})[j] + \delta \tag{5.7}$$

The robustness constraints are encoded as bounds on the input and *negation* of conditions on the output:

$$\forall j \in [1, d_0] \cdot \hat{x}[j] - \epsilon \leq x[j] \leq \hat{x}[j] + \epsilon \tag{5.8}$$
$$\forall j \neq \ell \in [1, d_L] \cdot \bigvee z^L[j] > z^L[\ell] \tag{5.9}$$

Theorem 5.3.1. If f is UNSAT, then E is (ϵ, δ)-robust at $\hat{x}$

Its correctness can be easily derived from Eq. (5.2) and the construction of f.

5.3.3 Solving the saliency-robustness problem by combining constraint-based NN verifiers with Jacobian bounding methods

In this part, we introduce a first cut to effectively verifying the saliency-robustness problem by combining two state-of-the-art neural network verification techniques – constraint-based neural network verifier and Jacobian bounding.

Given that the saliency-robustness problem can be encoded as a satisfiability problem over LRA, it could be solved by any off-the-shelf SMT solver such as Z3 [34] or CVC5 [9]. However, like the robustness problem, which can also be encoded as a satisfiability problem over LRA, solving the encoding using an off-the-shelf SMT solver hardly scale to any network of interesting size [69].

Adapting constraint-based NN verifiers for solving the saliency-robustness problem In this chapter, we use MARABOU [70], a dedicated state-of-the-art constraint-based NN verifier as the core solver. MARABOU extends the Simplex [94] algorithm used in linear programming with special mechanisms to handle ReLU activation function. Like Simplex, at each iteration, Marabou tries to fix a variable so that it doesn't violate its constraints. If in Simplex, a violation can only happen when a variable becomes out-of-bound, in Marabou a violation can also happen when a variable doesn't satisfy its activation constraints, thus MARABOU extends Simplex's pivot rules with a *PivotForRelu* rule and introduces *splitting* into the solving loop. Most importantly, MARABOU supports *disjunctions*, thus allowing it to express and solve more complicated verification specifications, compared to other tools like CROWN that only verifies the robustness property.

Out of the box, MARABOU can solve the satisfiability of $\phi_F \wedge \phi_P$ (which is exactly the robustness property). Since MARABOU only supports disjunctions but not implications, and doesn't have strict inequalities, to encode ϕ_G we model Eq. (5.5) and Eq. (5.6) using disjunction as follows:

$$z^i[j] \leq 0 \vee \partial z^i[j] = \partial h^i[j] \tag{5.10}$$

$$z^i[j] \geq 10^{-6} \vee \partial z^i[j] = 0 \tag{5.11}$$

Note that in Eq. (5.11) we use a small number to model strict inequality. This is a standard technique and is also recommended by MARABOU's developer [1]. Encoding $\phi_{\approx}$ in MARABOU is similar to encoding ϕ_P: we simply set the bounds for each of the entries of ∂x.

Precomputing Jacobian bounds It is not enough to encode the saliency-robustness problem into the form that MARABOU accepts. The core solving loop of MARABOU requires that every variable in the input has to be bounded. Thus, one of the first preprocessing steps in MARABOU is to derive bounds for all variables. Unfortunately, while MARABOU implements many procedures to derive and tighten bounds during the preprocessing phase, those procedures cannot compute bounds over disjunctions. In practice, that means we must find a way to effectively bound ∂h^i and ∂z^i and set them in MARABOU manually. Put it simply, we need to compute the Jacobian bounds for ∂h^i and ∂z^i

Bounding Jacobian is a hard and open question [147, 114, 67], and we do not attempt to solve the problem in this chapter. Instead, we use CROWN [114] – a state-of-the-art recursive algorithm to precompute the Jacobian bounds to use with MARABOU. To

[1] https://github.com/NeuralNetworkVerification/Marabou/issues/496

optimize memory usage, CROWN does not maintain the intermediate Jacobian bounds for all layers. We work around this issues by calling CROWN L times, each time marking one layer as the last layer in the computation graph, thus allowing us to collect the Jacobian bounds for all L layers in the network.[2]

5.4 Evaluation

In this section, we evaluate the saliency-robustness of the Vanilla Gradient, as well as compare the performance between Z3 – a state-of-the-art SMT solver, and our proposed method. We also conduct an experiment showing the relationship between the quality of the Jacobian bounds and the solving performance.

	Z3	MARABOU + CROWN		
Region	$\delta = 0.0001$	$\delta = 0.0001$	$\delta = 0.0005$	$\delta = 0.001$
$B(x_1, 0.05)$	TIMEOUT	3m46s UNSAT	TIMEOUT	TIMEOUT
$B(x_2, 0.03)$	TIMEOUT	1m34s UNSAT	1m22s UNSAT	5m22s UNSAT
$B(x_2, 0.05)$	TIMEOUT	TIMEOUT	TIMEOUT	TIMEOUT
$B(x_3, 0.03)$	TIMEOUT	ERROR	ERROR	ERROR
$B(x_3, 0.05)$	TIMEOUT	TIMEOUT	TIMEOUT	TIMEOUT
$B(x_4, 0.03)$	TIMEOUT	ERROR	ERROR	ERROR
$B(x_4, 0.05)$	TIMEOUT	1m19s UNSAT	1m28s UNSAT	3m23s UNSAT
$B(x_5, 0.05)$	TIMEOUT	ERROR	ERROR	ERROR
$B(x_6, 0.05)$	TIMEOUT	1m44s UNSAT	TIMEOUT	1m50s UNSAT
$B(x_7, 0.05)$	TIMEOUT	TIMEOUT	TIMEOUT	TIMEOUT

Table 5.2: Verifying the saliency-robustness property using Z3 and our method at different δs.

5.4.1 Experiment setup

Our experiments are based on our synthetic dataset for the five-arm bandit problem and benchmark from VNNCOMP23 [91] – the annual neural network verification competition. We use the MNIST dataset and the pre-trained model `mnistfc_256x2`, a 2-layers fully

[2]This is currently the recommended solution suggested by the developers, see https://github.com/Verified-Intelligence/auto_LiRPA/issues/46

connected network with 256 neurons for each layer. Due to the scalability of both CROWN in computing Jacobian bounds and MARABOU in solving, experiments with CNNs or bigger fully connected networks with bigger δs all result in TIMEOUT [3]. Note that by adding extra variables to represent gradients into MARABOU, every network is *double* in size, i.e., a query verifying the saliency-robustness of a 4-layer network has the same number of variables and constraints as verifying the robustness property of an 8-layer network.

Experiments are run on a `c5a.16xlarge` EC2 instances with 64 cores and 124GB of RAM. On all benchmarks and in both Z3 and MARABOU+CROWN, we allow the solver to use up to 30 cores. The timeout for each query is set to 10 minutes. Unless specified otherwise, we use the Crown-Optimized method in CROWN, and set the number of refinement iterations to 200 instead of the default value of 20. We call this the Reference config.

5.4.2 The saliency-robustness for the five-arm bandits over the whole input domain

We train BanditNet, a 2 layers FCN with 6 neurons each, on our synthetic benchmark. For BanditNet, we verify the saliency-robustness for the whole input domain ($\epsilon = 1$), at different values of δ ranging from 0.5 to 1.5. We run each query using both Z3 and MARABOU+CROWN, and results are summarized in Table 5.1. Z3 performs well on this small network, but even here, we observe a significant difference in performance between our method and Z3, across all δs, for both SAT and UNSAT queries. Interestingly, we also observe that the query becomes hardest near the δ border 1.25.

5.4.3 The saliency-robustness for `mnistfc_256x2` in known unsafe regions

To verify the usefulness of Vanilla Gradient as an explanation for the prediction of `mnistfc_256x2`, we look at inputs in the benchmarks that are known to have adversarial examples in their vicinity. If checking the saliency-robustness in the same vicinity returns UNSAT, we can claim that the Vanilla Gradient is a useful tool to explain the prediction in that region.

In the VNNCOMP23 benchmarks, MARABOU can find adversarial examples in 10 regions centered at 7 inputs at 2 different epsilon values (Table 5.2). As expected, Z3 does not scale to this network, while our method can verify 4 out of 10 regions at $\delta = 0.0001$.

[3] CROWN needs at least 40GB of GPU memory to bound the Jacobian of a 2-layer CNN network with only 8 channels and kernels of size 3×3

We also observe that as we increase δ, our queries become increasingly harder, resulting in more TIMEOUTs. There is an outlier at region $B(x_6, 0.05)$ in which at $\delta = 0.0005$ the query is timed out but at a harder $\delta = 0.001$ it can be solved again. It is interesting that other than returning TIMEOUT or SAT/UNSAT, we also observe cases where our method crashes the solver. Given the limited amount of time, we do not have a clear idea of the root cause of the crashes and we leave investigating those issues for future work.

5.4.4 The effect of bound's tightness on performance

	Reference config	No optimization		20 refinement iterations	
Region	Result	Result	avg ×	Result	avg ×
$B(x_1, 0.05)$	3m46s UNSAT	TIMEOUT	2.131	5m21s UNSAT	1
$B(x_2, 0.03)$	1m34s UNSAT	1m11s UNSAT	1.11	1m6s UNSAT	1.014
$B(x_2, 0.05)$	TIMEOUT	TIMEOUT	1.06	TIMEOUT	1
$B(x_3, 0.03)$	ERROR	ERROR	-	ERROR	-
$B(x_3, 0.05)$	TIMEOUT	TIMEOUT	1.05	TIMEOUT	1.002
$B(x_4, 0.03)$	ERROR	ERROR	1.69	ERROR	1
$B(x_4, 0.05)$	1m19s UNSAT	1m44s UNSAT	1.04	1m12s UNSAT	1
$B(x_5, 0.05)$	ERROR	ERROR	1.13	ERROR	1
$B(x_6, 0.05)$	1m44s UNSAT	TIMEOUT	1.04	1m44s UNSAT	1.004
$B(x_7, 0.05)$	TIMEOUT	TIMEOUT	1.07	TIMEOUT	1.004

Table 5.3: The effect of Jacobian bounds on solving the saliency-robustness problem. We use $\delta = 0.0001$ for this experiment. We show the average increase in Jacobian bounds in the two "avg ×" columns. The "avg ×" values for $B(x_3, 0.03)$ are missing since this query crashes CROWN, thus no bounds were computed.

As a new area of research, the quality of Jacobian bounds is improved rapidly, and in some cases, newer methods like CROWN can produce bounds orders of magnitudes smaller than older methods [114]. To get an idea of how big of a difference different Jacobian bounds can make on our method, we conduct an experiment in which we turn off all optimizations in CROWN, and another experiment in which we keep the same set of optimizations but use the default number of refinement iterations (20) to obtain looser bounds, then set them in MARABOU. Table 5.3 shows the solving result as well as the average increase in the size of the obtained bounds. Without any optimization, we can see that the obtained Jacobian bounds are quite loose (more than 2 times bigger compared

with the optimized bounds), resulting in more TIMEOUTs. In general, running Crown-Optimized for 20 iterations gives us relatively tight bounds, and the solving performance stays quite consistent between using 20 iterations and 200 iterations.

5.5 Conclusion

In this chapter, we propose a novel verification problem, called *saliency-robustness*, which aims to verify whether a Vanilla Gradient saliency map can serve as an explanation or certification for a prediction. We model the problem as a constraint satisfiability problem over linear real arithmetic and show that for small networks, our formulation can be solved by off-the-shelf SMT solvers like Z3. Furthermore, when Z3 doesn't scale to networks of bigger sizes, we propose a method combining constraint-based neural network verifier with Jacobian bounding to solve it more effectively. Experiments show that our method outperforms Z3 and scales to the `mnistfc_256x2` pre-trained network used in VNNCOMP23.

Limitations and Future Work There are several limitations in this chapter and open problems for future work. First, one thread to validity is the soundness of floating point arithmetic, which is a known issue in neural network verification in general [69]. Second, our proposed method is limited in scalability by both components: the Jacobian bounding algorithm of Crown does not scale to deeper neural networks and the quality of the bounds degrades significantly as we go deeper [114], and Marabou (or any constraint-based NN verifiers for that matter) is inherently slower than abstraction-based methods in exchange for being precise. Finally, our current method does not handle non-linear formulations of saliency maps, which are used in many works such as Integrated Gradients or SmoothGrad. Extending our work to support such saliency map functions is a challenge for future work.

As the first work at verifying the saliency-robustness property, our proposed method serves as a proof of concept, and we humbly hope we interest other researchers to build upon it toward more robust saliency maps.

The content of this chapter is adapted from the following work:

- Nham Le, Arie Gurfinkel, Xujie Si, Chuqin Geng. Towards Reliable Saliency Maps. Under submission at The 16th Asian Conference on Machine Learning. 2024, Hanoi, Vietnam.

Chapter 6

Verifying the Robustness Between the Latent Space, the Probe, and the Downstream Tasks in Interpreting Large Language Models

Large language models (LLMs) are surprisingly capable at a range of tasks, even when they are only trained on a simple "next-token" prediction task. However, how does this performance emerge out of the seemingly simple learning objective is still a mystery: do LLMs build a high-level structure representation of the world, or are they stochastic parrots? Recent researches in synthetic settings (the board game Othello) give evidence supporting the formal: it is possible to train a simple probe to reconstruct the board state from the latent space of a LLM, and the output of the downstream task can also be controlled by navigating said latent space such that the probe reports a different board state. In this chapter, we investigate the question of whether a *causality* relationship between the three entities: the latent space, the probe, and the task-solving head of the LLMs, exists.

6.1 Introduction

Large-language models (LLMs) have emerged as formidable tools in many tasks, surpassing human-being and revolutionizing various fields with their remarkable capabilities. Their prowess in tasks ranging from language translation to code generation, seemingly just from learning how to predict the next token, has surprised the community at large.

Yet, the underlying mechanisms that enable their exceptional performance remain somewhat enigmatic. While LLM's performance is undeniable, the intricacies of how they achieve it still elude full comprehension. Recent studies [77], however, provide compelling evidence suggesting that LLMs not only process data but also learn intricate representations of the world within the statistical patterns of the information they are trained on. This revelation opens new avenues for understanding the inner workings of these powerful models and underscores their potential for advancing artificial intelligence.

In the pioneering work [77], the authors argue for the existence of a world model inside LLMs, by looking at the connection between the three entities: the *Latent Space*, the *Trained Probe*, and the *Task Head* of the network. Their method – *Mechanical Intepretation* – starts with training an almost perfect GPT model $\mathcal{N}$ to predict either the next legal moves or the next best moves given a sequence of Othello moves, then train a probe $\mathcal{P}$ to reconstruct the board state given the hidden values h of $\mathcal{N}$. Note that at no point is the spatial information of the Othello board (e.g C4 is next to C5) given to $\mathcal{N}$: if exists in the model, it must have learned it by itself. It is fascinating that the probe, without any prior knowledge, can almost perfectly reconstruct the board in its 2D arrangement from the latent space. The authors go even further, by showing a causal relationship between the three entities $\mathcal{N}$, $\mathcal{P}$, h: given an Othello board state B, when moving in the latent space to trick the trained probe to reconstruct a different board B', the head will output legal or best moves corresponding to the board B' as well (instead of B).

In this chapter, we aim to use *Formal verification* to establish a mathematical certification for this connection. We exploit the fact that both the probes must be lightweight by construction: probing methods want to avoid the trap of training a second model explaining the data instead of explaining the original model [77], and the task-heads of the network usually are two- or one-layer fully connected networks; thus making them interesting targets for verification.

In this work, we make the following contributions:

- We introduce Trinity-robustness: a novel suite of safety properties designed to verify not only the three individual entities but, more importantly, the connections between them.
- We present a technique to encode the model and verify these properties as a set of first-order logic constraints, which are then solved using Marabou [70], a state-of-the-art neural network verifier.
- Our experimental results reveal that both the probes and the heads exhibit brittleness with respect to the latent space. Crucially, fixing one entity does not enhance the

robustness of the other, suggesting that the connection between the three entities is weaker than initially assumed.

6.2 The Trinity-robustness problem

Background for this chapter (Transformer architecture, Neural network verifiers, the board game Othello, and Mechanical interpretation) are presented in Chapter 2. We start this section by introducing our notations.

Denote s a sequence of moves on an Othello board resulting in a real-world board state $\mathcal{M}(s)$. Denote $h(s)$ the hidden state (vector v in Fig. 6.1) that is used for both the probe $\mathcal{P}$ and the task head $\mathcal{F}$ of the network. $\mathcal{P}$ produces a tensor of shape 60×3, corresponding to the probability of each of the 60 cells being black, white, or empty. In this chapter, we focus on the head network that outputs the set of next legal moves: $\mathcal{F}$ outputs a vector of length 60, in which a negative value means the tile is an illegal move and a positive value means the tile is a legal move.

We define $\approx_{\mathcal{M}}$ an approximation relation between the real-world model $\mathcal{M}(s)$ and the model reconstructed by $\mathcal{P}$. The real-world board state $\mathcal{M}(s)$ is a vector of size 60, in which each entry is either 0 (black), 1 (white), or 2 (empty), based on the rule of the game. We define

$$\mathcal{P}(h(s)) \approx_{\mathcal{M}} \mathcal{M}(s) \iff \forall i \in 1..60 \cdot argmax\mathcal{P}(h(s))[i] = \mathcal{M}(s)[i] \tag{6.1}$$

Additionally, we define the relation $\approx_{\mathcal{F}}$

$$\mathcal{F}(v_1) \approx_{\mathcal{F}} \mathcal{F}(v_2) \iff sign\mathcal{F}(v_1) = sign\mathcal{F}(v_2) \tag{6.2}$$

Mechanical interpretation uses probes to explain the inner state of the learned models, thus we argue for using formal verification to establish more confidence in the robustness of the probes: if we cannot trust the probes, how can we trust our interpretation of its output? At the same time, we also argue that while common LLMs are too large for the current state-of-the-art neural network verifiers, the heads in those LLMs are more lightweight and should be verified. Thus, we propose the following two robustness properties for the probes and the heads:

Given a perturbation norm ϵ, we define

- Head-robustness:

$$\forall \Delta \cdot ||\Delta||_\infty \leq \epsilon \implies \mathcal{F}(h(s) + \Delta) \approx_{\mathcal{F}} \mathcal{F}(h(s)) \tag{6.3}$$

- Probe-robustness:

$$\forall \Delta \cdot ||\Delta||_\infty \leq \epsilon \implies \mathcal{P}(h(s) + \Delta) \approx_{\mathcal{M}} \mathcal{M}(s) \tag{6.4}$$

While checking for the robustness of the probes and the heads are important, in this chapter, we also want to explore the connection between them: can the output of the head be used to *explain* the output of the probe, and vice versa? Thus, we propose two additional properties:

- Head proves probe ($\mathcal{F} \implies \mathcal{P}$):

$$\forall \Delta \cdot \mathcal{F}(h(s) + \Delta) \approx_{\mathcal{F}} \mathcal{F}(h(s)) \implies \mathcal{P}(h(s) + \Delta) \approx_{\mathcal{M}} \mathcal{M}(s) \tag{6.5}$$

- Probe proves head ($\mathcal{P} \implies \mathcal{F}$):

$$\forall \Delta \cdot \mathcal{P}(h(s) + \Delta) \approx_{\mathcal{M}} \mathcal{M}(s) \implies \mathcal{F}(h(s) + \Delta) \approx_{\mathcal{F}} \mathcal{F}(h(s)) \tag{6.6}$$

Together, the four specifications form the TRINITY-*robustness* problems. In the next section, we discuss how we encode and solve them. When the context is clear, we omit s.

6.3 Solving the Trinity-robustness with Marabou

In this section, we focus on how to solve the TRINITY-robustness properties against the pretrained probes and head in [77]. However, the technique presented here is generic and can be applied to probes and heads of other LLMs with minimal or no changes.

In [77], both the probes and the task heads consist of only linear layers and piecewise-linear activation functions. Thus, we show that TRINITY-robustness can be encoded into a satisfiability problem over linear real arithmetic (LRA). We first show that by constructing the encoding for probe-robustness, since they contain many of the needed techniques to construct all four TRINITY-robustness properties.

Encoding the probe-robustness property First, the property can be written as the following first-order logic formula f:

$$f := (\phi_h \wedge \phi_{\text{probe}}) \implies \phi_{\text{probe-robust}} \tag{6.7}$$

$$\text{in which} \tag{6.8}$$

$$\phi_h = \text{Constraints encoding the perturbation on the hidden vector } h \tag{6.9}$$

$$\phi_{\text{probe}} = \text{Constraints encoding the probe network} \tag{6.10}$$

$$\phi_{\text{probe-robust}} = \text{Constraints encoding the robustness properties} \tag{6.11}$$

First, we apply the same conversion as in software verification and model checking: to check for the validity of $A \implies B$, we check if there exists a satisfying assignment for $A \wedge \neg B$. Thus, we check SAT for

$$Q = (\phi_h \wedge \phi_{\text{probe}}) \wedge \neg\phi_{\text{probe-robust}} \tag{6.12}$$

To encode the perturbations $\forall \Delta \cdot ||\Delta||_\infty \leq \epsilon$, for each index i in h, we set its upper and lower bounds based on the true value of h (denote $\hat{h}$) as follows:

$$\phi_h = \bigwedge_i (h[i] > \hat{h}[i] - \epsilon) \wedge (h[i] < \hat{h}[i] + \epsilon) \tag{6.13}$$

To encode the network: Both the probe $\mathcal{P}$ and the head $\mathcal{F}$ can be encoded using the same encoding used by MARABOU [70]: for a linear layer $z^i = \boldsymbol{W}^i v^{i-1} + \boldsymbol{b}^i$, we have the constraint $z^i[j] = \sum_{k=1}^{|v_{i-1}|} \boldsymbol{W}^i[j][k] + \boldsymbol{b}^i[j]$ for each entry in the resulting layer z^i; and for a ReLU activation layer $v^i = ReLU(z^i)$, each entry in the result vector can be encoded using two implications:

$$z^i[j] > 0 \implies v^i[j] = z^i[j]$$
$$z^i[j] \leq 0 \implies v^i[j] = 0$$

For the probe-robustness property, we encode Eq. (6.1) as follows: At the tile i, the board has the true value $\mathcal{M}[i]$, then

$$\phi_{\text{probe-robust}} = \bigwedge_i \mathcal{P}(h)[i][\mathcal{M}[i]] = \max \mathcal{P}(h)[i] \tag{6.14}$$

$$\text{or} \tag{6.15}$$

$$\phi_{\text{probe-robust}} = \bigwedge_i \bigwedge_{j \neq \mathcal{M}[i]} \mathcal{P}(h)[i][\mathcal{M}[i]] > \mathcal{P}(h)[i][j] \tag{6.16}$$

Thus, by expanding the negation of a conjunction into a disjunction, we can encode $\neg\phi_{\text{probe-robust}}$ as:

$$\neg\phi_{\text{probe-robust}} = \bigvee_i \bigvee_{j \neq \mathcal{M}[i]} \mathcal{P}(h)[i][\mathcal{M}[i]] < \mathcal{P}(h)[i][j] \tag{6.17}$$

Altogether, we check for the probe-robustness by checking SAT for

$$Q = (\phi_h \wedge \phi_{\text{probe}})(\bigvee_i \bigvee_{j \neq \mathcal{M}[i]} \mathcal{P}(h)[i][\mathcal{M}[i]] < \mathcal{P}(h)[i][j]) \tag{6.18}$$

$$\text{or } Q = \bigvee_i \bigvee_{j \neq \mathcal{M}[i]} \phi_h \wedge \phi_{\text{probe}} \wedge \mathcal{P}(h)[i][\mathcal{M}[i]] < \mathcal{P}(h)[i][j] \tag{6.19}$$

While both Eq. (6.18) and Eq. (6.19) can be parsed and solved using MARABOU– our neural verifier of choice, the latter format allows us to explicitly parallelize the workload by creating 120 independent queries for MARABOU.

Encoding the head-robustness property , the only difference is in encoding $\phi_{\text{probe-robust}}$, thus the final check SAT query is

$$Q = \bigvee_i \phi_h \wedge \phi_{\text{head}} \wedge sign\mathcal{F}(h)[i] \neq sign\mathcal{F}(\hat{h})[i] \tag{6.20}$$

Encoding the $\mathcal{F} \implies \mathcal{P}$ and $\mathcal{P} \implies \mathcal{F}$ properties Due to their symmetry, we discuss only the encoding for $\mathcal{F} \implies \mathcal{P}$. $\mathcal{F} \implies \mathcal{P}$ can be written as the following formula:

$$f := (\phi_h \wedge \phi_{\text{head}} \wedge \phi_{\text{probe}} \wedge \phi_{\text{head-robust}}) \implies \phi_{\text{probe-robust}} \tag{6.21}$$

$$\text{in which} \tag{6.22}$$

$$\phi_h = \text{Constraints encoding the perturbation on the hidden vector } h \tag{6.23}$$

$$\phi_{\text{head}} = \text{Constraints encoding the head} \tag{6.24}$$

$$\phi_{\text{probe}} = \text{Constraints encoding the probe} \tag{6.25}$$

$$\phi_{\text{head-robust}} = \text{Constraints encoding the head-robustness} \tag{6.26}$$

$$\phi_{\text{probe-robust}} = \text{Constraints encoding the probe-robustness} \tag{6.27}$$

Using the same encoding scheme described above, the validity of $\mathcal{F} \implies \mathcal{P}$ can be answered by checking SAT for

$$Q = \bigvee_i \bigvee_{j \neq \mathcal{M}[i]} \phi_h \wedge \phi_{\text{probe}} \wedge \phi_{\text{head}} \wedge \underbrace{sign\mathcal{F}(h)[i] = sign\mathcal{F}(\hat{h})[i]}_{\phi_{\text{head-robust}}} \wedge \underbrace{\mathcal{P}(h)[i][\mathcal{M}[i]] < \mathcal{P}(h)[i][j]}_{\neg\phi_{\text{probe-robust}}} \tag{6.28}$$

Property	Number of sub-queries for each game
Head-robust	180
Probe-robust	384
$\mathcal{P} \implies \mathcal{F}$	180
$\mathcal{F} \implies \mathcal{P}$	384
Total	112800 (100 games)

Table 6.1: The number of input queries to MARABOU

6.4 Experiments

We evaluate our methods using a randomly generated set of Othello game sequences. Starting with an initial set of 1000 games, we exclude those where either the head makes an incorrect prediction or the probe incorrectly reconstructs the board. This filtering process results in a benchmark set of 617 game sequences, with lengths ranging from 5 to 30 moves. From this filtered set, we randomly select 100 games to serve as our benchmarks. We use the pre-trained head and probe extracted from [77], where the head is a linear layer and the probe is a 2-layer fully connected network. Both networks take the last hidden value of the Transformer block (post-LayerNorm) as the input. This input vector is of size 512.

For each game, we check all four of the TRINITY-robustness properties. Table 6.1 details the number of queries sent to MARABOU after splitting each disjunct into a set of queries. For each sub-query, we use MARABOU with the default parameters, setting a timeout of 5 minutes. The experiments are conducted on a Microsoft Azure Virtual Machine equipped with 64 cores and 256 GB of memory. Each property is tested at three different values of ϵ: 0.025, 0.05 and 0.1.

RQ1: Are the trained task head and probe robust with respect to perturbation in the latent space? Table 6.2 presents the results of verifying the robustness of the head and the probe. The head demonstrates exceptional robustness across all tested epsilon values: there are no adversarial points in the latent space for 99.45% of the queries at $\epsilon = 0.025$, and even at $\epsilon = 0.1$, MARABOU can only find a counter-example in 10.14% of the queries. Additionally, none of the head-robustness queries result in a timeout. This can be attributed to its relatively simple architecture: the head is just a linear layer of affine transformation.

The probe shows robustness at epsilon values of 0.025 and 0.05, with 98.39% and

ϵ	Head-robust			Probe-robust		
	SAT	UNSAT	T/o	SAT	UNSAT	T/o
0.025	0.55%	99.45%	0%	0.88%	98.39%	0.73%
0.05	1.53%	98.46%	0%	6.48%	81.07%	12.45%
0.1	10.14%	89.86%	0%	43.26%	7.88%	48.86%

Table 6.2: Verifying the head- and probe-robustness

ϵ	$\mathcal{F} \implies \mathcal{P}$			$\mathcal{P} \implies \mathcal{F}$		
	SAT	UNSAT	T/o	SAT	UNSAT	T/o
0.025	0.55%	99.45%	0%	0.84%	98.09%	1.06%
0.05	1.53%	98.46%	0%	6.42%	80.31%	13.26%
0.1	10.1%	89.9%	0%	43.11%	7.64%	49.25%

Table 6.3: Verifying the $\mathcal{P} \implies \mathcal{F}$ and $\mathcal{F} \implies \mathcal{P}$

81.07% of the queries returning UNSAT, respectively. However, the difficulty of the queries significantly increases as epsilon is scaled up: less than 1% of the queries timeout at $\epsilon = 0.025$, but this number rises to 12.45% at $\epsilon = 0.05$ and nearly half (48.86%) at $\epsilon = 0.1$. Notably, at $\epsilon = 0.1$, MARABOU is unable to prove the robustness of the probe in most cases, with less than 10% of the queries being proven UNSAT within the time limit.

RQ2: If the perturbation does not change the output of the probe (head), is the head (probe) robust? The results, as detailed in Table 6.3, reveal some unexpected findings. Fixing the output of the probe does not enhance the robustness of the head. Likewise, fixing the output of the head does not significantly improve the robustness of the probe, but makes the queries more challenging, leading to an increase in the number of timeouts across all three epsilon values. This is surprising, as it suggests that the causal connection between the probe and the head is not as strong as [77] proposed: there exist multiple points in the latent space where the output of the head and the probe disagree with each other, thus the output of one cannot be used to explain the output of the other.

Discussion We hypothesize two possible reasons for this intriguing phenomenon. First, the meaning of ϵ in the latent space is not well-defined. While adding 0.05 or 0.1 to a red channel of a pixel has a clear visual interpretation, moving along a dimension in the latent space a value of 0.05 does not have a clear real-world equivalence. Thus, our choice of ϵ might be too large or small. Second, even if our perturbation range is meaningful, it

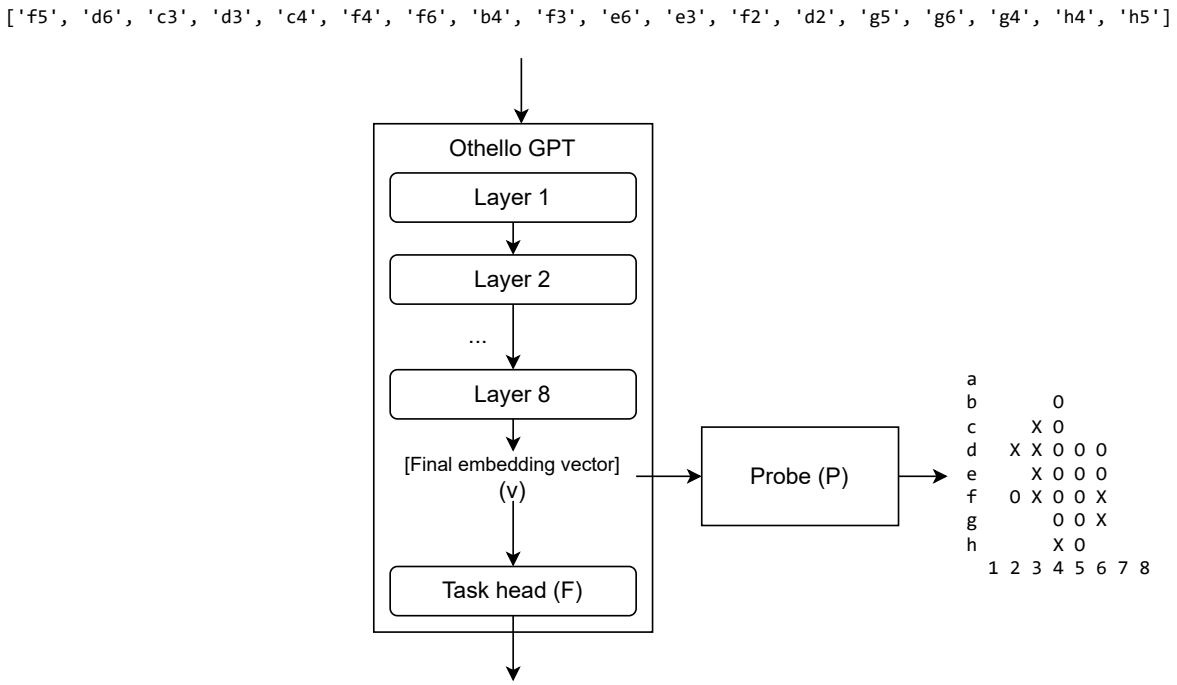
is possible that the adversarial examples where the head and the probe disagree do not correspond to any real input sequence s. Thus, the head and probe may not prove each other's robustness with respect to the *continuous* latent space, but the causal relation with respect to the *discrete* input space may still hold. Reversing a point in the latent space to a corresponding input or the closest one is an interesting future work.

6.5 Conclusion

In this chapter, we attempt to provide a mathematical proof for the connection between the latent space, the task head, and the probe used in Mechanical Interpretation. Specifically, we investigate whether the output of the head can be used to prove the robustness of the probe and vice versa. While empirical evidence from [77] suggests a causal connection between the probe and the head, our technique using formal verification reveals points in the latent space where the head and probe outputs disagree. Surprisingly, we find these discrepancies across a substantial region of the randomly sampled latent space, indicating that the connection between the three entities is weaker than initially thought. We hope that these findings will encourage further research into Mechanical Interpretation.

The content of this chapter is adapted from the following work:

- Nham Le, Henry Guo, Arie Gurfinkel, Xujie Si, Chuqin Geng. TRINITY-robustness: Verifying the Causality Between the Latent Space, the Probe, and the Downstream Tasks in Large Language Models. Submitted to at FMCAD'24 Student Forum.




```
output by running F(v):
 tensor([-2.0611, -2.8018, -3.0063,  7.7758,  7.8361, -1.6718, -2.3594, -2.6148,
         -2.2483, -2.5150, -2.5595, -1.9102,  7.7744, -1.5704, -2.2384, -3.5364,
         -2.2118, -2.2511, -1.9897, -2.8324,  7.7565,  7.8481, -1.6253, -2.7198,
         -2.4036, -2.8010, -2.3921,                   -2.3050,  7.7830, -2.2417,
         -1.4258, -2.7268, -2.0945,                   -2.7569,  7.7833, -2.8513,
          7.8171, -2.2584, -1.8082, -2.5208, -2.7762, -2.5894, -2.6374, -2.4938,
          7.8177, -1.1532,  7.7892, -2.4260, -2.8173, -2.5380, -1.3652, -3.0821,
         -2.2105, -2.0345, -0.3020, -1.9035, -2.4227,  7.8137, -1.4604, -2.2261])
```

Figure 6.1: OthelloGPT predicting the next legal moves given a sequence of play. The output of the head in this case is a vector of size 60, encoding all cells in the board. The vector is reformatted for clarity, with moves predicted as legal by the head highlighted.

Chapter 7

Future Work

As machine learning finds increasing application across diverse industries, explaining and validating the operational mechanisms of neural networks assumes heightened significance. This dissertation investigates the integration of explanation and verification techniques to enhance understanding and bolster confidence in neural networks. Additionally, we explore the development of an efficient distributed Satisfiability Modulo Theories (SMT) solver – the heart of many neural network verifiers. We outline a number of future directions on both fronts: using verification to enhance explanation methodologies; and strategies for further optimizing the scalability of distributed SMT solvers.

7.1 On Robust Explanations for Neural Networks

Robust training for explanation methods Recent advancements in the verification of robustness properties of neural networks have facilitated the development of robust training methodologies. In these methodologies, robustness is integrated into the training phase of the neural network, rather than being assessed post hoc [4]. For explanation methods that rely on training an auxiliary network to interpret the behavior of the primary network (e.g. probing), we propose that incorporating robust training techniques into the auxiliary networks can enhance their reliability. This, in turn, increases our confidence in the interpretations derived from it.

Handling non QF-LRA constraints In Chapter 3, Chapter 5, and Chapter 6, we work with explanation functions that can be directly encoded as QF-LRA constraints.

This allows us to use off-the-shelf neural verifiers with a minimal amount of changes in the solvers, but also limits what explanations we can verify. Being able to handle complex activation functions such as *tanh* or statistical procedures like *sampling* will enable us to work with richer specifications.

7.2 On Efficient Distributed Solving

A unified framework for clause-sharing and partitioning for distributed SMT The two most common methods to scale SMT solvers are partitioning [64, 136] and clause-sharing [34]. In this dissertation, we present our results on building a clause-sharing based distributed solver based on cvc5. One natural extension is to combine both partitioning and clause-sharing. To the best of our knowledge, SMTS [84] is the only solver that attempts this combination, by sharing clauses between multiple solver copies solving the same partition.

Learning clause sharing heuristics As argued in Chapter 4, one crucial piece of making an efficient clause-sharing solver is to choose which clause to share. In Chapter 4, we utilize a rather simple heuristic based on the size of the learned clause, and while CVC5-D shows great results, we believe that a better heuristics can improve its performance even further.

Recent advances in the realm of SAT solving have witnessed the application of machine learning techniques, specifically Graph Neural Networks (GNNs), to effectively rank learned clauses [134, 145]. These developments highlight the feasibility of integrating machine learning into the selection process of SMT-solving strategies. Given that cvc5 relies fundamentally on a SAT solver, integrating a learned clause scoring mechanism adapted from SAT could potentially offer a significant performance boost to CVC5-D. Moreover, the incorporation of sophisticated structures within theory clauses holds promise for even greater gains in efficiency.

Enhancing the clause-selection process through advanced machine learning models not only aligns with recent trends in SAT solving but also opens avenues for exploring novel methodologies in SMT solving. By harnessing the power of machine learning to optimize clause sharing, we anticipate not only improving the efficiency of CVC5-D but also advancing the broader capabilities of SMT solvers in handling complex problem domains effectively.

Chapter 8

Conclusion

This thesis presents a new paradigm for safe and explainable AI by leveraging the strengths of both explanation and verification methods to validate neural networks. Explanation methods are intuitive but lack rigorous mathematical backing, whereas verification methods are robust by design but limited in their ability to validate large input spaces. By verifying explanation methods, we address these issues: enhancing the trustworthiness of explanations and significantly expanding the verifiable input space.

Combining explanation and verification opens a new research dimension, presenting novel properties to be checked, new challenges for tool developers, and innovative ways to design explanation methods. In this thesis, we address fundamental issues with the commonly used ϵ-robustness specification, advocating for its augmentation with explanation functions. To demonstrate the feasibility and effectiveness of our approach, we propose and develop techniques to augment ϵ-robustness with some of the most widely used explanation methods. Specifically, we focus on Saliency Maps for Computer Vision and Mechanical Interpretation for Large Language Models, enhancing the interpretability and trustworthiness of these AI models. Moreover, we develop advanced techniques to solve the proposed specifications using state-of-the-art tools, either individually or in combination. Recognizing the inevitable scalability issues associated with verification methods, we discuss our efforts in building a distributed Satisfiability Modulo Theories (SMT) solver – the backbone of many neural network verifiers.

This thesis demonstrates the potential of combining explanation and verification for safe and explainable AI. Looking forward, this approach can be extended not only to other explanation functions but also to the creation of new explanation methods. Explanation methods should be verifiable to ensure trust in their outputs. Furthermore, our success in

building a distributed SMT solver holds significant potential for improving neural verifiers. We hope that this work can be incorporated into future verifiers to address more complex properties in larger networks.

In conclusion, this thesis presents a comprehensive approach to combining explanation and verification for safe and explainable AI. By addressing the limitations of both methods and leveraging their strengths, we enhance the reliability and interpretability of neural networks. Our work lays the foundation for future research in this area, offering new directions and opportunities for the development of more robust and trustworthy AI systems.

References

[1] Saliency maps in Tensorflow 2.0. https://usmanr149.github.io/urmlblog/cnn/2020/05/01/Salincy-Maps.html. [Accessed 15-Mar-2023].

[2] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 9525–9536, Red Hook, NY, USA, 2018. Curran Associates Inc.

[3] Aws Albarghouthi. Introduction to neural network verification. *CoRR*, abs/2109.10317, 2021.

[4] Brendon G. Anderson, Tanmay Gautam, and Somayeh Sojoudi. An overview and prospective outlook on robust training and certification of machine learning models, 2022.

[5] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller. How to explain individual classification decisions. *J. Mach. Learn. Res.*, 11:1803–1831, aug 2010.

[6] Stanley Bak and Parasara Sridhar Duggirala. Simulation-equivalent reachability of large linear systems with inputs. In Rupak Majumdar and Viktor Kunčak, editors, *Computer Aided Verification*, pages 401–420, Cham, 2017. Springer International Publishing.

[7] Stanley Bak, Changliu Liu, and Taylor T. Johnson. The second international verification of neural networks competition (VNN-COMP 2021): Summary and results. *CoRR*, abs/2109.00498, 2021.

[8] Tomás Balyo, Peter Sanders, and Carsten Sinz. Hordesat: A massively parallel portfolio SAT solver. *CoRR*, abs/1505.03340, 2015.

[9] Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. cvc5: A versatile and industrial-strength SMT solver. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, volume 13243 of *Lecture Notes in Computer Science*, pages 415–442. Springer, 2022.

[10] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2016.

[11] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB Standard: Version 2.6. Technical report, Department of Computer Science, The University of Iowa, 2017. Available at `www.SMT-LIB.org`.

[12] Clark Barrett, Roberto Sebastiani, Sanjit Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability, Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, chapter 33, pages 825–885. IOS Press, February 2021.

[13] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya Nori, and Antonio Criminisi. Measuring neural net robustness with constraints, 2016.

[14] Yoshua Bengio and Yann LeCun. Scaling learning algorithms towards AI. In *Large Scale Kernel Machines*. MIT Press, 2007.

[15] A. Biere, A. Biere, M. Heule, H. van Maaren, and T. Walsh. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. IOS Press, NLD, 2009.

[16] Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without bdds. In W. Rance Cleaveland, editor, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 193–207, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[17] Christian Borch. Machine learning, knowledge risk, and principal-agent problems in automated trading. *Technology in Society*, 68:101852, 2022.

[18] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020.

[19] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[20] B. Buchberger. A theoretical basis for the reduction of polynomials to canonical forms. *SIGSAM Bull.*, 10(3):19–29, August 1976.

[21] Alex Bäuerle, Daniel Jönsson, and Timo Ropinski. Neural activation patterns (naps): Visual explainability of learned concepts, 2022.

[22] John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-8:679 – 698, 12 1986.

[23] Nicholas Carlini and David Wagner. *Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods*, page 3–14. Association for Computing Machinery, New York, NY, USA, 2017.

[24] Chih-Hong Cheng, Georg Nührenberg, and Hirotoshi Yasuoka. Runtime monitoring neuron activation patterns. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 300–303. IEEE, 2019.

[25] Sohee Cho, Wonjoon Chang, Ginkyeng Lee, and Jaesik Choi. Interpreting internal activation patterns in deep temporal neural networks by finding prototypes. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 158–166, 2021.

[26] Wahid Chrabakh and Rich Wolski. Gridsat: A chaff-based distributed sat solver for the grid. pages 37– 37, 12 2003.

[27] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *International Conference on Computer Aided Verification*, pages 154–169. Springer, 2000.

[28] 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*, pages 238–252, 1977.

[29] David A. Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Springer Publishing Company, Incorporated, 4th edition, 2015.

[30] Zihang Dai, Hanxiao Liu, Quoc V Le, and Mingxing Tan. Coatnet: Marrying convolution and attention for all data sizes. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 3965–3977. Curran Associates, Inc., 2021.

[31] Nilesh Dalvi, Pedro Domingos, Mausam, Sumit Sanghai, and Deepak Verma. Adversarial classification. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, page 99–108, New York, NY, USA, 2004. Association for Computing Machinery.

[32] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, jul 1962.

[33] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, jul 1960.

[34] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[35] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[36] Thomas G. Dietterich and Eric Horvitz. Rise of concerns about AI: reflections and directions. *Commun. ACM*, 58(10):38–40, 2015.

[37] Bruno Dutertre. Yices 2.2. In Armin Biere and Roderick Bloem, editors, *Computer-Aided Verification (CAV'2014)*, volume 8559 of *Lecture Notes in Computer Science*, pages 737–744. Springer, July 2014.

[38] Ömer Faruk Ereken and Cigdem Tarhan. Breast cancer detection using convolutional neural networks. In *2022 International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, pages 597–601, 2022.

[39] D. Erhan, Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. 2009.

[40] Nicholas Frosst and Geoffrey E. Hinton. Distilling a neural network into a soft decision tree. *ArXiv*, abs/1711.09784, 2017.

[41] Joachim von zur Gathen and Jrgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, USA, 3rd edition, 2013.

[42] Keith O. Geddes, Stephen R. Czapor, and George Labahn. *Algorithms for Computer Algebra*. Kluwer Academic Publishers, USA, 1992.

[43] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 3–18, 2018.

[44] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016.

[45] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Commun. ACM*, 63(11):139–144, oct 2020.

[46] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2014.

[47] Bryce Goodman and Seth Flaxman. European union regulations on algorithmic decision-making and a "right to explanation". *AI Magazine*, 38(3):50–57, oct 2017.

[48] Google. grpc.io. https://grpc.io/. [Accessed 15-Mar-2023].

[49] A. Gurfinkel, Temesghen Kahsai, Anvesh Komuravelli, and Jorge A. Navas. The seahorn verification framework. In *International Conference on Computer Aided Verification*, 2015.

[50] Youssef Hamadi and Lakhdar Sais, editors. *Handbook of Parallel Constraint Reasoning*. Springer, 2018.

[51] Boris Hanin and David Rolnick. Complexity of linear regions in deep networks. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 2596–2604. PMLR, 2019.

[52] Boris Hanin and David Rolnick. Deep relu networks have surprisingly few activation patterns. In *NeurIPS*, pages 359–368, 2019.

[53] Matthias Hein and Maksym Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[54] Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In *International Conference on Theory and Applications of Satisfiability Testing*, 2016.

[55] Marijn J. H. Heule, Oliver Kullmann, Siert Wieringa, and Armin Biere. Cube and conquer: Guiding cdcl sat solvers by lookaheads. In *Proceedings of the 7th International Haifa Verification Conference on Hardware and Software: Verification and Testing*, HVC'11, page 50–65, Berlin, Heidelberg, 2011. Springer-Verlag.

[56] Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

[57] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *CoRR*, abs/2006.11239, 2020.

[58] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, nov 1997.

[59] Boming Huang, Yuxiang Huan, Li Da Xu, Lirong Zheng, and Zhuo Zou. Automated trading systems statistical and machine learning methods and hardware implementation: a survey. *Enterprise Information Systems*, 13(1):132–144, 2019.

[60] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Computer Science Review*, 37:100270, 2020.

[61] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. *CoRR*, abs/1610.06940, 2016.

[62] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In Rupak Majumdar and Viktor Kunčak, editors, *Computer Aided Verification*, pages 3–29, Cham, 2017. Springer International Publishing.

[63] Antti E. J. Hyvärinen, Tommi Junttila, and Ilkka Niemelä. A distribution method for solving SAT in grids. In Armin Biere and Carla P. Gomes, editors, *Theory and Applications of Satisfiability Testing - SAT 2006*, pages 430–435, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[64] Antti E. J. Hyvärinen, Matteo Marescotti, Leonardo Alt, and Natasha Sharygina. Opensmt2: An smt solver for multi-core and cloud computing. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing – SAT 2016*, pages 547–553, Cham, 2016. Springer International Publishing.

[65] brian ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander T Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown, Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Nikhil J. Joshi, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. Do as i can, not as i say: Grounding language in robotic affordances. In Karen Liu, Dana Kulic, and Jeff Ichnowski, editors, *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 287–318. PMLR, 14–18 Dec 2023.

[66] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 448–456. JMLR.org, 2015.

[67] Matt Jordan and Alexandros G Dimakis. Exactly computing the local lipschitz constant of relu networks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 7344–7353. Curran Associates, Inc., 2020.

[68] Kyle D. Julian, Jessica Lopez, Jeffrey S. Brush, Michael P. Owen, and Mykel J. Kochenderfer. Policy compression for aircraft collision avoidance systems. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, pages 1–10, 2016.

[69] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In Rupak Majumdar and Viktor Kunčak, editors, *Computer Aided Verification*, pages 97–117, Cham, 2017. Springer International Publishing.

[70] Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljic, David L. Dill, Mykel J. Kochenderfer, and Clark W. Barrett. The marabou framework for verification and analysis of deep neural networks. In *CAV (1)*, volume 11561 of *Lecture Notes in Computer Science*, pages 443–452. Springer, 2019.

[71] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, and Rory Sayres. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). 2017.

[72] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, October 2014. Association for Computational Linguistics.

[73] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1885–1894. JMLR.org, 2017.

[74] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[75] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[76] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[77] Kenneth Li, Aspen K. Hopkins, David Bau, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Emergent world representations: Exploring a sequence model trained on a synthetic task, 2023.

[78] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[79] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*, 2017.

[80] Zachary C. Lipton. The mythos of model interpretability, 2016.

[81] Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher Strong, Clark Barrett, and Mykel J. Kochenderfer. Algorithms for verifying deep neural networks. *Foundations and Trends® in Optimization*, 4(3-4):244–404, 2021.

[82] Wentao Liu, Hanglei Hu, Jie Zhou, Yuyang Ding, Junsong Li, Jiayi Zeng, Mengliang He, Qin Chen, Bo Jiang, Aimin Zhou, and Liang He. Mathematical language models: A survey, 2024.

[83] Matteo Marescotti, Antti E. J. Hyvärinen, and Natasha Sharygina. Clause sharing and partitioning for cloud-based smt solving. In Cyrille Artho, Axel Legay, and Doron Peled, editors, *Automated Technology for Verification and Analysis*, pages 428–443, Cham, 2016. Springer International Publishing.

[84] Matteo Marescotti, Antti E. J. Hyvärinen, and Natasha Sharygina. SMTS: distributed, visualized constraint solving. In Gilles Barthe, Geoff Sutcliffe, and Margus Veanes, editors, *LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Awassa, Ethiopia, 16-21 November 2018*, volume 57 of *EPiC Series in Computing*, pages 534–542. EasyChair, 2018.

[85] J.P. Marques-Silva and K.A. Sakallah. Grasp: a search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[86] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations, 2017.

[87] Christoph Molnar. *Interpretable Machine Learning*. 2 edition, 2022.

[88] Christoph Molnar. *Interpretable Machine Learning*. 2 edition, 2022.

[89] Max Moroz. *AFL Quick Start Guide*. Google.

[90] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: engineering an efficient sat solver. In *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, pages 530–535, 2001.

[91] Mark Niklas Müller, Christopher Brix, Stanley Bak, Changliu Liu, and Taylor T. Johnson. The third international verification of neural networks competition (vnncomp 2022): Summary and results, 2023.

[92] Mark Niklas Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel, and Martin Vechev. Prima: Precise and general neural network certification via multi-neuron convex relaxations, 2021.

[93] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814. Omnipress, 2010.

[94] John A. Nelder and Roger Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[95] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving sat and sat modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll(t). *J. ACM*, 53(6):937–977, nov 2006.

[96] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *CoRR*, abs/1811.03378, 2018.

[97] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *CoRR*, abs/1511.08458, 2015.

[98] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[99] Luca Pulina and Armando Tacchella. Challenging smt solvers to verify neural networks. *AI Commun.*, 25(2):117–135, apr 2012.

[100] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

[101] 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, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144, 2016.

[102] H. G. Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, 74(2):358–366, 1953.

[103] Henry Gordon Rice. *Rice's theorem.* Wikipedia.

[104] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2024.

[105] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[106] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[107] Dominik Schreiber and Peter Sanders. Scalable sat solving in the cloud. In Chu-Min Li and Felip Manyà, editors, *Theory and Applications of Satisfiability Testing – SAT 2021*, pages 518–534, Cham, 2021. Springer International Publishing.

[108] Roberto Sebastiani and Silvia Tomasi. Optimization modulo theories with linear rational costs. *ACM Transactions on Computational Logic*, 16, 10 2014.

[109] Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *CoRR*, abs/1610.02391, 2016.

[110] François Serre, Christoph Müller, Gagandeep Singh, Markus Püschel, and Martin Vechev. Scaling polyhedral neural network verification on GPUs. In *Proc. Machine Learning and Systems (MLSys)*, 2021.

[111] Sanjit A. Seshia, Dorsa Sadigh, and S. Shankar Sastry. Toward verified artificial intelligence. *Commun. ACM*, 65(7):46–55, jun 2022.

[112] Dhruv Shah, Błażej Osiński, brian ichter, and Sergey Levine. Lm-nav: Robotic navigation with large pre-trained models of language, vision, and action. In Karen Liu, Dana Kulic, and Jeff Ichnowski, editors, *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 492–504. PMLR, 14–18 Dec 2023.

[113] Alex Sherstinsky. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *CoRR*, abs/1808.03314, 2018.

[114] Zhouxing Shi, Yihan Wang, Huan Zhang, J. Zico Kolter, and Cho-Jui Hsieh. Efficiently computing local lipschitz constants of neural networks via bound propagation. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 2350–2364. Curran Associates, Inc., 2022.

[115] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 3145–3153. JMLR.org, 2017.

[116] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[117] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Workshop Track Proceedings*, 2014.

[118] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Workshop Track Proceedings*, 2014.

[119] Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin Vechev. Beyond the single neuron convex barrier for neural network certification. In *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.

[120] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[121] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 3(POPL), jan 2019.

[122] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda B. Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *CoRR*, abs/1706.03825, 2017.

[123] Bernardo Subercaseaux and Marijn J. H. Heule. The packing chromatic number of the infinite square grid is at least 14. In *International Conference on Theory and Applications of Satisfiability Testing*, 2022.

[124] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. *CoRR*, abs/1703.01365, 2017.

[125] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

[126] Ellen Tobback and David Martens. Retail credit scoring using fine-grained payment data. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 182(4):1227–1246, 2019.

[127] Hoang-Dung Tran, Stanley Bak, Weiming Xiang, and Taylor T. Johnson. Verification of deep convolutional neural networks using imagestars. In *Computer Aided Verification: 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21–24, 2020, Proceedings, Part I*, page 18–42, Berlin, Heidelberg, 2020. Springer-Verlag.

[128] Hoang-Dung Tran, Neelanjana Pal, Patrick Musau, Diego Manzanas Lopez, Nathaniel Hamilton, Xiaodong Yang, Stanley Bak, and Taylor T. Johnson. Robustness verification of semantic segmentation neural networks using relaxed reachability. In *Computer Aided Verification: 33rd International Conference, CAV 2021, Virtual Event, July 20–23, 2021, Proceedings, Part I*, page 263–286, Berlin, Heidelberg, 2021. Springer-Verlag.

[129] G. S. Tseitin. *On the Complexity of Derivation in Propositional Calculus*, pages 466–483. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983.

[130] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, pages 5998–6008, 2017.

[131] Linda Wang, Zhong Qiu Lin, and Alexander Wong. Covid-net: A tailored deep convolutional neural network design for detection of covid-19 cases from chest x-ray images. *Scientific Reports*, 10(1):1–12, 2020.

[132] Longwei Wang, Chengfei Wang, Yupeng Li Li, and Rui Wang. Explaining the behavior of neuron activations in deep neural networks. *Ad Hoc Networks*, 2021.

[133] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. Beta-CROWN: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. *Advances in Neural Information Processing Systems*, 34, 2021.

[134] Wenxi Wang, Yang Hu, Mohit Tiwari, Sarfraz Khurshid, Kenneth McMillan, and Risto Miikkulainen. Neuroback: Improving CDCL SAT solving using graph neural networks. In *The Twelfth International Conference on Learning Representations*, 2024.

[135] Tjark Weber, Sylvain Conchon, David Déharbe, Matthias Heizmann, Aina Niemetz, and Giles Reger. The SMT competition 2015-2018. *J. Satisf. Boolean Model. Comput.*, 11(1):221–259, 2019.

[136] Amalee Wilson, Andres Nötzli, Andrew Reynolds, Byron Cook, Cesare Tinelli, and Clark W. Barrett. Partitioning strategies for distributed SMT solving. In Alexander Nadel and Kristin Yvonne Rozier, editors, *Formal Methods in Computer-Aided Design, FMCAD 2023, Ames, IA, USA, October 24-27, 2023*, pages 199–208. IEEE, 2023.

[137] Jeannette M. Wing. A specifier's introduction to formal methods. *Computer*, 23(9):8–24, 1990.

[138] Jeannette M. Wing. Trustworthy AI. *Commun. ACM*, 64(10):64–71, 2021.

[139] Christoph M. Wintersteiger, Youssef Hamadi, and Leonardo de Moura. A concurrent portfolio approach to SMT solving. In Ahmed Bouajjani and Oded Maler, editors, *Computer Aided Verification*, pages 715–720, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[140] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, January 2021.

[141] Han Xu, Yao Ma, Haochen Liu, Debayan Deb, Hui Liu, Jiliang Tang, and Anil K. Jain. Adversarial attacks and defenses in images, graphs and text: A review. *International Journal of Automation and Computing*, 17:151–178, 2020.

[142] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. SATzilla: Portfolio-based algorithm selection for SAT. *J. Artif. Intell. Res.*, 32:565–606, 2008.

[143] Yichen Xu and Yanqiao Zhu. A survey on pretrained language models for neural code intelligence, 2022.

[144] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization, 2015.

[145] Boyu Yu, Hao Chen, Tsz Hang Ng, Xiaolong Ma, and Bei Yu. Neuroselect: Learning to select clauses in sat solvers. In *Proceedings of the 61st ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2024.

[146] Michal Zalewski. *AFL Whitepaper*. Google.

[147] Huan Zhang, Pengchuan Zhang, and Cho-Jui Hsieh. Recurjac: An efficient recursive algorithm for bounding jacobian matrix of neural networks and its applications. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019.

[148] Quanshi Zhang, Yu Yang, Haotian Ma, and Ying Nian Wu. Interpreting cnns via decision trees. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6254–6263, 2019.

[149] Xuming Zhang and Mirella Lapata. Sentence simplification with deep reinforcement learning. *arXiv preprint arXiv:1703.10931*, 2017.

[150] Luisa M Zintgraf, Taco S Cohen, Tameem Adel, and Max Welling. Visualizing deep neural network decisions: Prediction difference analysis. *arXiv preprint arXiv:1702.04595*, 2017.