ThanhVu Nguyen's Annotated NSF CAREER'23 Proposal

This document is inspired by Philip Guo's NSF Annotated CAREER'18 proposal (no longer available online). So I am doing the same here. I try to maintain the exact content and format of the submitted proposal though I have redacted various personal info and names (like this ■).

Timeline: Start writing in May (though already been working on the topic for almost a year), finish a week early and get feedback, submit in late July, received notification from NSF PM in mid Dec (seems quite early, I didn't expect it), and after some budget adjustment the award was official in Feb'23.

Background This proposal was on the verification of neural networks. It was a completely new research direction for me and was also my 3rd (last) try. So it was quite risky but I felt strongly about the topic and have solid preliminary results (described later).

Advice Before getting into the details, here are some general advice.

1. **Contact PMs** to make sure proposal get to the *right panel*. NSF PMs are friendly and want junior faculty to succeed. Send them the proposal summary and discuss with them to see if the topic would fit their panels (and customize the contents for such panels).

   My research in software analysis has three possible PMs, whose panels can be very different.

2. Have **prelim work** (e.g., prototype, small experiments, pilot study, workshop/short papers). Both the reviewers *and* you will have more confidence about your proposed work. This is especially important if you try to work on a new research direction or topic.

3. **Finish early** to get reviews. Fresh eyes will see issues and provide useful advice. For example, my original draft didn't have several things that I believe important including secret source in intro and concrete examples that you will see later.

   For example, industrial support shows the work has practical impacts. Education/outreach appears stronger if others are willing to help.

4. Get relevant **supporting letters**. Just as with prelim work, this increases the confidence that you can do the work and that the work has broad impacts. The **letter from the Chair** is also important and read by the reviewers.

5. If possible, get **your students** involved and help. My students and I have worked on various pieces of the proposal (e.g., prototypes, experiments) and so they help provide figures and data to support the proposed ideas. They can also check correctness of various technical parts and provide suggestions.

   I have only 1 short paper related to the topic but spent lots of effort in developing prototype and obtain prelim results to support the proposed work.

6. It is **OK** to work on a new research direction, as long as you can show you can do it. This might be risky and controversial (e.g., a sound advice is that reviewers should not be surprised when they read what your CAREER is about). But it could be done, especially if you no longer want to work on something you have been working on for almost 10 years!

What would I do different? Not too much. Of course after working on the project for almost a year I understand the problem and challenges a lot more and would make various changes to the proposal. And obviously I would also use ChatGPT to assess and improve the writing. But these are wishful thoughts; I really felt I have already tried my best when the proposal was submitted. :)

Overview

Deep Neural Networks (DNNs) have emerged as an effective approach to tackling real-world problems. However, just like traditional software, DNNs can have "bugs" and be attacked. This naturally raises the question of how DNNs should be tested, validated, and ultimately *verified* to meet the requirements of relevant robustness and safety standards.

To address this question, researchers have developed powerful formal methods and tools to verify DNNs. However, despite many recent advances, these approaches and tools still have challenges in achieving good precision and scalability, could produce unsound results, and do not apply to DNNs such as Graph Neural Networks (GNNs). To address these challenges, this proposal will develop techniques for accurate and scalable DNN verification, stress-testing DNN verifiers and certifying proved results, and tackling GNNs through the lenses of other DNN approaches.

NSF program managers look at the summary and determine if it would go to their bucket and consequently their panels. Reviewers also look at the summary to determine if they want to review proposal. Very similarly to an abstract in a paper.
I sent a draft summary to PM A, who thought it might fit better with PM B, who referred me to another PM C, who indeed became the PM of the proposal. So, in short, use the summary and contact PMs.

Intellectual Merit

The project has four research components (RCs).

RC#0 develops NeuralSAT, a constraint-solver for verifying DNNs that combines the conflict-driven clause learning ability of modern SAT solving and an abstraction-based theory solver in SMT solving and DNN verification. We already have a prototype for NeuralSAT that works with Feedforward Neural Networks (FNNs), and preliminary results show that the prototype is several orders of magnitude faster the state of the art in constraint-based DNN verification.

RC#1 makes NeuralSAT more precise and efficient at scale. Key insights include building on the non-convex *weak max-plus* abstraction developed in our prior invariant work and exploiting heuristics and optimizations that make SAT solving successful, e.g., branching heuristics and parallelization.

This might be a bit too detail for summary. Though I felt that it helped the PMs to determine that this proposal does/does not fit their panels.

RC#2 helps developers find bugs in their DNN verifiers during production and certify their results during deployment. We will exploit clause learning to bring the ability to certify the results of NeuralSAT (e.g., proofs of proved results) and build upon successful metamorphic testing techniques that found bugs in mature SMT solvers and compilers to stress test NeuralSAT and other DNN verifiers.

RC#3 explores GNNs, a powerful model in deep learning but with few existing formal techniques and tools, potentially because of their different and complex structures and behaviors. To analyze GNNs, our insight is using influential substructures of input graphs to an GNN to reduce that GNN to an FNN, allowing for the applications of FNN analyzers to GNNs. Our pilot study shows the potential and an interesting application of the approach—we were able to *infer rich properties* of GNNs using an off-the-shelf invariant generation tool for FNNs.

Broader Impacts

The research will benefit society by improving the reliability of systems embedding DNNs. The research contributes to ML by developing effective techniques to verify DNNs, allowing AI/ML researchers and users to improve their DNNs and deploy them with confidence. The findings from this project will be used to develop new courses and a "DNN Verification by Examples" book. The research will support graduate and undergraduate student researchers and outreach activities for K-12 students in Prince William county of Northern Virginia.

A preview of my Broader Impacts section §2 in the description. I also mention a bit about education integration and outreach here as this summary of the entire proposal.

Keywords deep neural networks, verification, formal methods, satisfiability checking, clause learning, resolution proofs, stress-testing, abstractions, reduction, feedforward and graph neural networks

Keywords might not be necessary but I include them anyway as they might help get the right reviewers.

1 Introduction

Deep Neural Networks (DNNs) have emerged as an effective approach to tackling real-world problems. Among many others, they have been used for image recognition [44, 66], autonomous driving [106, 113], power grid control [121], fake news detection [130], drug synthesis and discovery [40], diabetes detection [43], and not surprisingly, COVID-19 detection and diagnosis [62, 102].

However, just like traditional software, DNNs can have "bugs", e.g., producing unexpected results on inputs that are different from those in training data, and be attacked, e.g., small perturbations to the inputs by a malicious adversary or even sensorial imperfections result in misclassification [50, 108,149,156,163]. These issues, which have been observed in many DNNs [36,127] and demonstrated in the real world [29], naturally raise the question of how DNNs should be tested, validated, and ultimately *verified* to meet the requirements of relevant robustness or safety standards [48, 60].

To address this question, researchers have developed powerful formal methods and tools to verify DNNs (e.g., [49, 61, 72, 85, 135, 140]). However, despite many recent advances, DNN verification still has many limitations, especially with scalability and precision (e.g., incorrectly claiming safe systems unsafe). Constraint-based approaches [28, 28, 49, 59, 61] aim to both correctly prove and disprove properties, but do not scale to large networks. In contrast, abstraction-based approaches [85, 123, 125, 139, 140] correctly prove valid properties but do not guarantee disproved results (can produce spurious counterexamples). Newer abstraction techniques can check counterexamples and refine their abstractions—but this makes them slow in disproving [9]. In general, however, abstraction-based approaches scale well. For instance, the top performers, e.g., β-CROWN [140] and nnenum [7], in the annual Neural Network Verification competitions (VNN-COMPs) all use abstractions [8].

Surprisingly, the problem of DNN verification, even with networks with nonlinear activation functions such as "ReLU", has been shown in the seminal Reluplex work [59] to be reducible from the classical satisfiability (SAT) problem [23]. However, while modern constraint solvers scale well to large formulae and help make the theoretically intractable SAT problem practically tractable [67], constraint-based DNN verifiers do not seem to scale and be competitive with the best DNN verifiers. For example, in our experiments (§4.1.2), Marabou [61], the successor of Reluplex and state-of-the-art constraint-based tool for DNN verification, consistently times out on large benchmarks and falls far behind abstraction-based tools.

Inspired by Reluplex and Marabou, but with the belief that constraint-based approaches can do much better, we propose to develop **NeuralSAT**, a new SAT solving approach to DNN verification that aims to both prove and disprove properties *and* do so efficiently. Our insight is to integrate *clause learning* [14,78,79] (to improve performance by addressing SAT challenges such as backtracking) in modern SAT solving with an *abstraction-based theory solver* (to quickly check for infeasibility) in SMT solving and abstraction-based DNN verification. We are working on a prototype for NeuralSAT that supports feedforward neural networks (FNNs), a major deep learning model, with the ReLU activation function. Preliminary results using the standard ACAS XU [56] benchmarks show that the prototype is at least *four orders of magnitude* faster than Marabou, which would often time out and return no solution. Initial investigation suggests that the synergistic combination of clause learning and abstraction solver is the "secret sauce" distinguishing NeuralSAT and helping it avoid the scalability problem faced by Marabou (and other constraint-based approaches).

Compared to nnenum, the winner of VNN-COMP'21 for FNNs, our prototype is slower in proving properties (but faster in disproving). Nonetheless, these results are encouraging, especially when both Marabou and nnenum are heavily optimized, e.g., both effectively leverage parallelization [19, 145]—optimizations that we will pursue in this research.

OK, the most important section, the intro! 2 pages to get the reviewers to like the proposal. If they don't get excited by then, they will find ways to reject the proposal. Similar to reviewing a paper.

Start with a **very high level** description and motivation of the problem. In general, put effort in to making the intro understood by *everyone*.

Motivate by talking about what people have done, limitations and challenges.

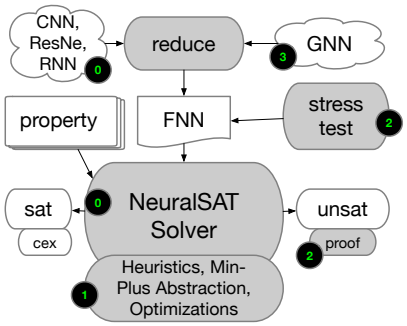
Now, talk about the proposed work. It's a good idea here to reveal the "secret ingredients" that allow the proposed research to overcome limitations of current work and tackle the problem.

Try to get to the proposed work on the first page of the proposal. Presenting the proposed work after the first page might be bit too late.

Mention prelimin work here to show preparation.

In addition to **DNN verification**, we will explore and develop techniques to tackle several challenging and interesting problems in the subfield of DNN analysis. These include **certifying proved results of verifiers**, **stress-testing verifiers**, and **reducing other DNNs to FNNs**. The SAT-based design allows NeuralSAT to leverage techniques in SAT solving to effectively tackle some difficult tasks, e.g., using clause learning to obtain resolution proofs for proved results. The reduction allows for the transfer of ideas and techniques among different DNNs. In sum, if successful, this research will set a new standard and open new research directions in DNN analyses.

1.1 Intellectual Merit



Intellectual Merit gives more details on the research components (RCs). Using a diagram to show the connection or integration of the proposed RCs.

The figure on the right depicts the proposed research, provided through four research components (RCs) shown with black ovals.

RC#0 is on developing the NeuralSAT framework, and what remains in RC#0 is to evaluate and extend the prototype tool to support other feedforward learning models such as Convolutional, Residual, and Recurrent Networks. **RC#1** will make NeuralSAT precise and efficient at scale. Key insights include building on the non-convex *weak max-plus* abstraction developed in our prior invariant generation work [92] and leveraging heuristics and optimizations that make SAT solving successful, e.g., branching heuristics and parallelization. *We aim to give NeuralSAT the power and impact in DNN verification the same way that modern constraint solvers such as Z3 [84] have in program analyses.*

Our experience shows that well-known and mature DNN verifiers can be unsound and prove invalid properties—a highly undesirable behavior for verifiers. Currently, few DNN verifiers have the ability to certify their (proved) results, making it difficult to trust them, especially in safety-critical situations where verifiers are needed. Key challenges include effectively deriving steps used by the verifier in producing proved results and finding bugs in already well-tested verifiers. To address these, **RC#2** exploits clause learning to certify the results of NeuralSAT (e.g., proofs of proved results), and applies successful metamorphic testing techniques that found thousands of bugs in mature SMT solvers and compilers [69,104,144] to stress test NeuralSAT and other DNN verifiers. *This research will help developers find bugs in their DNN verifiers during production and certify their results during deployment.*

An easy to understand *ending sentence* for each RC to summarize the RC.

Graph Neural Networks [116] (GNNs) are another powerful model in deep learning, but have few formal analyses or tools, potentially because they have different structures and behaviors compared to FNNs (e.g., the inputs to a GNN are *graphs*). However, our preliminary work [98] shows that we can create FNNs from influential substructures of input graphs to GNNs, thus allowing the applications of FFN analyzers to GNNs. Our pilot study (§4.4.2), in which we manually apply the idea to a small GNN simulating the BFS (Breadth First Search) algorithm, shows the potential and an interesting application of the approach—we were able to *infer rich properties* of BFS using an off-the-shelf invariant generation tool for FNNs. **RC#3** will explore GNNs through the lenses of analyses developed for FNNs. *As with any new research problem with few existing solutions, we have to start somewhere, and reducing it to something we already know seems to be worthwhile and even promising as FNN techniques and tools become more powerful.*

Finish the main part of the Intro within the first 2 pages!

1.2 PI's Qualifications and Preparations

Here's where to convince the reviewers that you are the right person to get the work done.

My research in software engineering and programming languages focus on invariant generation [51, 70, 86, 89, 91, 96] and program analyses (verification, synthesis, and repair) [68, 77, 97, 141, 159, 160].

Our dynamic invariant generation work DIG [88] was recognized with the Distinguished Paper award at ICSE'14 [90], and our program repair work GenProg was recognized with the Impact Paper Award at GECCO'19 [30] and 10-year Most Influential Paper Award at ICSE'19 [141].

Talk about how prelim work and background would help achieve each individual RCs. AI verification is a new research direction for me and so prelim work is important to show I am prepared.

To prepare for this project, my students and I have developed the NeuralSAT prototype and obtained preliminary results (§4.1.2) to demonstrate the approach and support the work in RCs #0–1. The weak max-plus abstraction in RC#2 will build on our prior dynamic invariant generation work [92, 96]. RC#3 will use our recent work [98] and pilot study on reducing GNNs to FNNs and experience in developing reduction techniques [97]. My experience in releasing research tools and contributing to large open-source projects (e.g., I wrote the initial Python API helper [128] for Z3 [84] and created the comprehensive nonlinear benchmark suite [129] for the annual Software Verification competitions [16]), will help with the management and distribution of tools and datasets.

The proposal has 3 letters of support from academic and industrial collaborators, who will help various part of the work. In general, having proper collaborative letters would strengthen the proposal.

The proposal includes collaborative letters from [redacted] who will help with optimizations in RC#1 and contribute to the DNN book (§5.1), [redacted], a [redacted] researcher who will help with industrial adoption and benchmarks (§2), and [redacted], a faculty in the CS department at GMU who will help with outreach activities (§5.2).

2 Broader Impacts

This required section gives a broader goal of your work. In other words, if successfull, what will the proposed work bring beyond the usual research/academic achievements?

As our world increasingly depends on ML to enhance our lives, the research will *benefit society* by improving the reliability of systems embedding DNNs. The research will also benefit researchers and developers who apply DNNs to safety-assured systems, e.g., autonomous driving [113] and aircraft collision control [56], and thus provide a path to improving the safety of the public.

In addition to contributing novel algorithms and tools to formal methods and software engineering, the research *contributes to AI/ML* by developing effective techniques to verify trained DNNs, allowing AI/ML researchers and users to improve their networks and deploy them with confidence. The research also connects different types of DNNs (e.g., FNNs and GNNs), allowing for the transfer of techniques and results among them. Similar to many DNN verification works [11, 85, 110, 114, 148, 153], we will share our findings with the AI/ML community.

The research will produce *industry-relevant tools*, and we will work to transfer technology to the industry. We have been collaborating with [redacted] on several projects [87, 99, 158], and they have expressed interest in NeuralSAT and can help with adoption and contribute real-world benchmarks.

A brief overview of educational and outreach activities here and referring the reader to its own section §5.2

The findings from this project will be used into my courses and ongoing mentoring and outreach activities. I will integrate research materials in an upcoming online software engineering course and an interactive book on DNN analyses. I have successfully involved undergraduate students in my research (e.g., [51, 52, 86, 87, 93–96]), and this project will provide support to continue these efforts and explore new opportunities, e.g., broadening participation to K-12 students as outlined in §5.2.

3 Background

For this specific proposal I have a background section to help the reviewers understand more about the problems and proposed work.

Sat Solving Modern SAT solvers employ the CDCL (Conflict-Driven Clause Learning) algorithm [14,78,79], which combines the classical DPLL [25] (Davis-Putnam-Logemann-Loveland) algorithm with *clause learning*. DPLL has three main components: assigning truth values to variables, inferring additional assignments, and analyzing conflicts and backtracking to a previous decision level to try new assignments. CDCL also has these DPLL components but extends conflict analysis to learn clauses, which help avoid conflicts and backtrack more effectively. State-of-the-art SMT (Satisfiability Modulo Theories) solvers such as Z3 [84] and CVC [12] employ the DPLL(T) algorithm [67, 100] to check the satisfiability of an SMT formula involving non-boolean variables, e.g., real numbers. DPLL(T) combines DPLL/CDCL with *theory solvers* to reason about formulae in

different theories, e.g., using a linear programming (LP) solver to analyze linear constraints.

DNNs [59] A *feed-forward neural network* (FNN), the quintessential model in deep learning [35], consists of an input layer, multiple hidden layers, and an output layer. Each layer has a number of neurons, each connected to neurons from the previous layer through a predefined set of weights (derived by training the network with data). A DNN is an FNN with at least two hidden layers.

The output of a DNN is obtained by iteratively computing the values of neurons in each layer. The value of a neuron in the input layer is the input data. The value of a neuron in the hidden layers is computed by applying an *affine transformation* to values of neurons in the previous layers, then followed by an *activation function* such as the popular Rectified Linear Unit (ReLU) activation. Specifically, the value of a hidden neuron y is $ReLU(w_1v_1 + \ldots + w_nv_n + b)$, where b is the bias parameter of y, $w_i, \ldots, w_n$ are the weights of y, $v_1, \ldots, v_n$ are the neuron values of the previous layer of y, $w_1v_1 + \cdots + w_nv_n + b$ is the affine transformation, and $ReLU(x) = \max(x, 0)$ is the ReLU activation. The value of a neuron in the output layer is evaluated similarly but can be without applying the activation function.

Given a DNN N and a property ϕ, the **DNN verification problem** asks if ϕ is a valid property of N. Typically, ϕ is a formula of the form $\phi_{in} \Rightarrow \phi_{out}$, where ϕ_{in} is the precondition over the inputs of N and ϕ_{out} is the postcondition over the outputs of N. This form of properties has been used to encode safety and security requirements of DNNs, e.g., safety specifications to avoid collision in unmanned aircraft [63] and *adversarial robustness* [60] properties desired by all DNNs, in which a small input perturbation does not cause major spikes in the DNN's outputs.

It starts to get dry and technical so I complement that with some high-level motivation, e.g., allowing us to avoid collision in unmanned aircraft.

A DNN verifier attempts to find a *counterexample* input satisfying ϕ_{in} but results in an output violating ϕ_{out}. If no such counterexample exists, ϕ is a valid property of N; otherwise, ϕ is not.

DNN verification can be encoded as an SMT formula and in fact is NP-Complete [59] (thus theoretically reducible to other NP-Complete problems, such as satisfiability solving). However, general SAT and SMT solvers do not scale for large and complex formulae encoding real-world DNNs containing many layers and neurons. Thus, for scalability, many techniques and tools are specifically developed for DNN verification (§4.6) by exploiting the structures of the DNNs and using various abstractions to approximate ReLU computations.

Example Fig. 1 shows a simple DNN with 2 inputs, 2 hidden neurons, and 1 output. The weights of a neuron are shown on the edges connecting to it, and the bias is shown above or below each neuron. The outputs of the hidden neurons are computed using affine transformation and ReLU, e.g., $x_3 = ReLU(-x_1 - 0.5x_2 - 1.0)$. The output neuron is computed with just the affine transformation, i.e., $x_5 = x_3 - x_4 - 1$.

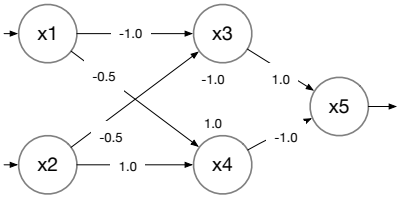


Fig. 1: A DNN with 2 inputs, 2 hidden neurons, and 1 output.

Having a small, concrete example can help illustrate the problem. Also use this example at various places in the proposal.

A valid property for this DNN is that the output is $x_5 \leq 0$ for any inputs $x_1 \in [-1, 1], x_2 \in [-2, 2]$. An invalid property is that $x_5 > 0$ for those similar inputs (a counterexample is $\{x_1 = 0.0, x_2 = -2.0\}$, from which the network evaluates to $x_5 = -1.0$). These properties can be used to check *safety requirements* (e.g., one rule in the aircraft collision system in [59, 63] is "if the intruder is distant and significantly slower than us, then we stay below a certain threshold") or *local robustness* [60] conditions (a form of adversarial robustness stating that if individual inputs are close within certain regions, then the output remains within some bound).

Showing how this toy example can represent something real and significant, e.g., safety violation in aircraft collision systems.

4 Proposed Research

Here comes the technical or "meat" of the proposal. The intro paragraph emphasizes that the proposed work is beneficial and applicable to general AI verification and also as an opportunity to recall the RCs.

The proposed NeuralSAT project consists of four research components (RCs). While we implement and evaluate the research using the NeuralSAT framework, the underlying principles are applicable to general DNN verification. For example, RC#0 shows the promise of combining clause learning and abstraction to verify DNNs; RC#1 develops the weak max-plus domain, which is usable for general abstraction-based DNN analyses; RC#2 explores resolution proofs for clause learning approaches and techniques to stress test off-the-shelf DNN verifiers; and RC#3 analyzes GNNs using existing DNN techniques and tools.

4.1 RC#0: NeuralSAT Solver (Prototype and Preliminary Results)

High level description. First remind the reviewers about the limitations and challenges, and then introduce the RC.

This RC0 is essentially about the NeuralSAT prototype that I have *already* developed and the proposed tasks are relatively minimal, e.g., completing the prototype and generalize it a bit.

State-of-the-art DNN verifiers are powerful and continuously improved. However, unlike the field of constraint solving that many of these techniques rely on, we still lack a scalable approach of modern SAT solving that renders the theoretically intractable SAT problem tractable in practice. This RC is on developing NeuralSAT, a new SAT-based approach to scalabble DNN verification. The design of NeuralSAT is inspired by the core algorithms used in SAT solvers such as clause learning and theory solving, and thus NeuralSAT can be considered as a satisfiability solver for DNNs and carries with it the power of modern SAT solvers.

4.1.1 Constraint-based Solving for DNN Verification

Fig. 2 gives an overview of NeuralSAT, which follows the DPLL(T) framework (§3) and consists of DPLL/CDCL components (light shades) and the theory solver (dark shade).

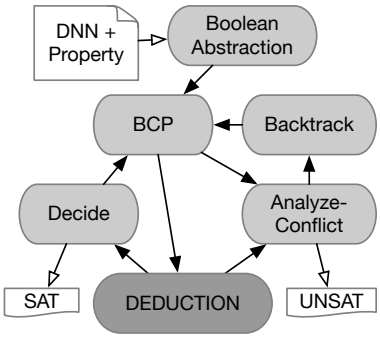


Fig. 2: NeuralSAT Overview.

The diagram allows non-experts to have a high-level idea of the proposed work.

First, NeuralSAT abstracts the DNN verification task into a SAT problem, consisting of only clauses over boolean variables (*Boolean Abstraction*). Here, NeuralSAT creates boolean variables to represent the *activation* status of neurons, e.g., with ReLU a (hidden) neuron is *active* if the input value to ReLU is positive and *inactive* otherwise. Next, NeuralSAT creates clauses asserting each neuron is either active/true or inactive/false. This abstraction allows NeuralSAT to use (i) DPLL/CDCL to search for truth values satisfying these clauses and (ii) the theory solver to check the feasibility of truth assignments with respect to the constraints encoding the DNN and the property of interest.

NeuralSAT now enters an iterative process to satisy the activation clauses. First, NeuralSAT assigns a truth value to an unassigned variable (*Decide*) and infers additional assignments caused by this assignment (*Boolean Constraint Propagation*). Next, NeuralSAT invokes the theory solver (*DEDUCTION*), which (i) tightens the bounds of the network inputs using the current assignments and an LP solver and (ii) abstracts (approximates) the bounds of the network outputs using the new input bounds, (iii) and checks if these bounds are feasible with the property of interest.

If the theory solver determines feasibility, NeuralSAT continues with new assignments (back to *Decide*). Otherwise, NeuralSAT analyzes the infeasibility (*Analyze-Conflict*) and learns clauses to avoid such infeasibility and backtrack to a previous iteration (*Backtrack*). This process repeats until NeuralSAT no longer can backtrack (returns `unsat`, indicating the DNN has the property) or finds a complete assignment for all activation variables (returns `sat`, and the user can query the solver for a counterexample).

	ϕ_1	ϕ_2^*	ϕ_3	ϕ_4	ϕ_5	ϕ_6	ϕ_7^*	ϕ_8^*	ϕ_9	ϕ_{10}
NeuralSAT	1025.36	**22.84**	526.77	330.83	83.51	127.35	262.01	**0.15**	142.00	191.99
Marabou	TO	821.41	8309.09	133.97	1259.74	250.41†	TO	TO	TO	3134.35†
nnenum	**168.08**	37.75	**48.67**	**44.02**	**3.83**	**23.60**	**232.87**	1.13	**9.86**	**2.55**

Tab. 1: Comparison to Marabou and nnenum. Runtimes are in seconds, **bold** means best time, TO means timeout (after 3 hours), ϕ^* means invalid properties to be disproved, † means incorrect results.

Example We demonstrate how NeuralSAT proves that the DNN in Fig. 1 has the safety/robustness property mentioned in §3: for all inputs $x_1 \in [-1, 1], x_2 \in [-2, 2]$, the DNN produces the output $x_5 \leq 0$. Thus, we want to show that *no* value assignments to x_1, x_2 satisfying the input properties that would result in $x_5 > 0$, the negation of the output property.

Using the example introduced in §3 to demonstrate NeuralSAT. In hindsight this level of details might not be needed. Though it might be useful for an expert to understand and appreciate the novelty of the proposed work.

First, NeuralSAT creates two variables v_3 and v_4 to represent the (pre-ReLU) activation status of the hidden neurons x_3 and x_4. For example, $v_3 = T$ means x_3 is `active`, i.e., $-x_1 - 0.5x_2 - 1 > 0$, and $v_3 = F$ means x_4 is `inactive`, i.e., $-x_1 - 0.5x_2 - 1 \leq 0$. Next, NeuralSAT forms two clauses $\{v_3 \vee \overline{v_3}\ ;\ v_4 \vee \overline{v_4}\}$ indicating these variables are either `active` or `inactive`.

Now, NeuralSAT searches for truth assignments for activation variables to satisfy the clauses. In iteration 1, from the input property $x_1 \in [-1, 1], x_2 \in [-2, 2]$ NeuralSAT uses the polytope [125] abstraction to approximate the upper bound $x_5 \leq 0.55$ and deduces that the output $x_5 > 0$ *might be* feasible, and then continues with a (random) decision $v_3 = F$. In iteration 2, NeuralSAT determines that, with the new constraint imposed by the assignment $v_3 = F$, the upper bound is $x_5 \leq 0$ and thus cannot satisfy $x_5 > 0$. It then analyzes the infeasibility to learn the clause v_3 (i.e., v_3 must be T) and backtracks to the previous decision $v_3 = F$ to erase it. In iteration 3, NeuralSAT infers $v_3 = T$ because of the recently learned clause v_3, determines that this assignment is feasible, and continues with a (random) decision $v_4 = T$. After 5 such iterations, NeuralSAT learns the clauses $\{v_3, \overline{v_3} \vee v_4, \overline{v_3} \vee \overline{v_4}\}$, realizes unsatisfiability, and returns unsat (i.e., the property is valid).

4.1.2 NeuralSAT Prototype and Preliminary Results

These preliminary results show that NeuralSAT is promising and competitive. It emphasizes that despite being still in development, the prototype is already extremely competitive. This RC provides necessary work to complete the tool and make it the *best* in DNN verification.

We are developing a prototype for NeuralSAT in Python that supports FNNs with ReLU. Tab. 1 compares the performance of the prototype to Marabou [61] and nnenum [7] using the standard ACAS (airborne collision avoidance system) XU [56] benchmarks, which have 45 networks and 10 desired safety properties[1]. We followed the setups for the FNNs with ReLU category in VNN-COMP'21, in which nnenum is the best performer while Marabou ranks 3rd. We used the run scripts and versions provided by Marabou and nnenum in VNN-COMP'21, and ran the experiment on a Linux machine with an Intel 3.6 GHz CPU with 16 threads and 32 GB RAM.

Other than ϕ_4, NeuralSAT outperformed Marabou in all cases, many of which were several orders of magnitude faster (e.g., 3 hrs vs. 0.15s for ϕ_8) and some of which caused Marabou to return incorrect results ($\phi_{6,10}$ and additional details in §4.3). This makes NeuralSAT the most effective constraint-based tool using the ACAS XU benchmarks. However, NeuralSAT ran slower than nnenum, especially when proving properties. Nonetheless, these results are encouraging because (i) our prototype is unoptimized while both Marabou and nnenum are, e.g., both leverage multicores in modern CPUs (the performance of nnenum dropped significantly when run with a single thread, e.g., 283s for ϕ_6 and 2543s for ϕ_8), and (ii) NeuralSAT appears to perform better than nnenum in disproving properties, suggesting that our approach is effective at finding counterexamples. Further investigation shows the synergy of abstraction-based theory solver, to quickly derive infeasibility,

[1]E.g., property ϕ_3 requires the airplane to turn sharp right if the intruder is near and approaching from the left.

and clause learning, to avoid infeasibility and guide future decisions, allows NeuralSAT to run fast, even as an unoptimized single-thread tool.

While being encouraged by these preliminary results, we do not yet understand the variability across the verification instances (e.g., the slow run time for ϕ_4). The rest of this subtask is to complete the prototype and evaluate the NeuralSAT's approach and performance more thoroughly.

The proposed task here is minimal, i.e., just finishing and bringing the prototype to light.

4.1.3 Supporting Other Feedforward Learning Models

We will also extend NeuralSAT to support other feedforward learning models, namely Convolutional (CNNs), Residual Learning (ResNets), and Recurrent (RNNs) Neural Networks, and activation functions such as sigmoid and tanh. These extensions allow us to evaluate the NeuralSAT approach on more general feedforward DNNs, and also help NeuralSAT become versatile and competitive with verifiers supporting these networks (e.g., many participants of VNN-COMPs also support CNNs and ResNets and non-ReLU activation functions).

While being used for a wide variety of tasks (and especially well-known for image and speaker recognition problems [35, 44, 54, 66]), ResNets, CNNs, and RNNs, are all feedforward learning models [35]. In an FNN, computation flows forward from the input layer, through each of the hidden layers, to the output layer without looping back. In contrast, a ResNet allows neurons to jump over some layers; an RNN allows neurons to connect those in the previous layer (thus creating a loop); and a CNN employs a *convolution* transformation, in which the value of a neuron is computed by a sliding window of neurons instead of a row of neurons in the previous layer as in FNNs.

NeuralSAT is built for general feedforward models, and thus will not require major changes to support these DNNs, especially ResNets and CNNs. For example, NeuralSAT still encodes the computations of these networks as linear constraints, and can use any connection flow the network has (specified in the open ONNX format [6]). For RNNs, which contain loops, we will use "unrolling" [1] to create an FNN to simulate the execution of the RNN for some fixed k iterations, or a more scalable approach of creating an FNN that overapproximates but has the same size as the RNN [54].

We will also support continuous activation functions such as sigmoid and tanh [8,35]. Recall that for ReLU, we use boolean variables to abstract and split the ReLU computation into two regions: active (> 0) and inactive (≤ 0). For a continuous function such as sigmoid or tanh, we can also split the computation into two regions (e.g., for tanh, we split at 0) or k regions for better precision (at the cost of creating more boolean variables and clauses to simulate non-binary values).

4.2 RC#1: Max-Plus Abstraction, Heuristics, and Optimizations

Again, in the intro part, provide a high level description, e.g., the aim of the RC.

This RC will improve the precision and scalability of NeuralSAT by (i) developing the *weak max-plus abstraction* to precisely and efficiently capture the non-convex behavior of activation functions, (ii) exploring heuristics that make SAT solvers powerful, and (iii) leveraging practical engineering improvements to optimize the implementation of NeuralSAT. This RC aims to make the NeuralSAT approach reach its full potential and become the top performer in upcoming VNN-COMPs and ultimately impactful to DNN analyses as SMT solvers such as Z3 to program reasoning.

4.2.1 The Weak Max-Plus Abstract Domain

DNN verification is challenging because activation functions such as ReLU produce non-convex regions, which are expensive to analyze. Abstraction-based techniques thus develop abstract domains, e.g., interval [139], zonotope [123], polytope [125], to overapproximate non-convex regions into convex ones for more efficient analyses. However, such an overapproximation, especially when being repeatedly applied in a large DNN, leads to imprecision, causing invalid counterexamples.

Existing work in program analysis [3,4] (including ours [58,92]) and abstraction-based DNN verification [38] has used the non-convex *max-plus* abstract domain [45] to represent complex program behaviors such as ReLU. However, just as with general polyhedral analysis, reasoning in (general) max-plus is prohibitively expensive and has a great impact on scalability [38, 92].

Instead of general max-plus, we will build on our prior work [92,96] on *weak max-plus* abstraction, a more restrictive form of general max-plus (analogously, an interval or octagon is a restricted form of a polygon). The first two pictures in Fig. 3 show the only two types of lines possible in weak max-plus and a weak max-plus "convex hull" (a tight overapproximation) over a set of points in 2D. Although weak max-plus is restrictive and produces a relatively peculiar non-convex shape, it can be computed efficiently [92] and, importantly, can accurately capture the non-convex region $y = \max(x, 0) \Rightarrow y = x \ \vee \ y = 0$ of ReLU (the upper-left line in the first picture in Fig. 3). The last picture in Fig. 3 shows how weak max-plus *exactly* captures ReLU (black line) while the popular abstract domains zonotope (blue) and polytope (orange) significantly overapproximate.

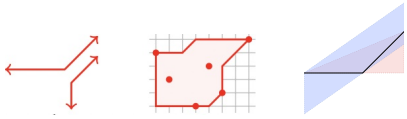


Fig. 3: Weak max-plus and comparison to zonotope and polytope.

As before, talk about existing work to show you have the right background to do this.

Our dynamic invariant work [92, 96] computes min max-plus convex hulls over trace points. Here, we develop weak-max plus to statically analyze DNNs. We will follow the process of creating abstraction domains from abstraction-based DNN work, e.g., [2, 32, 125]. This consists of creating transfer (abstraction) functions to manipulate weak max-plus regions over DNN computations (e.g., affine transformations and ReLU), convert from interval constraints over inputs to weak max-plus, and convert the weak max-plus result back to intervals. We will adapt general max-plus abstraction work (e.g., [3, 38]) to weak max-plus where the main modifications would be fixing the number of variables and coefficients, which are restrictions that weak max-plus has over general max-plus.

We will integrate the new weak max-plus abstraction into the theory solver of NeuralSAT, which currently uses polytope. Weak max plus can also be used as a stand-alone abstraction domain and adopted by existing abstraction-based DNN tools (e.g., nnenum uses several abstraction domains).

4.2.2 SAT Solving Heuristics

Modern SAT/SMT solvers are powerful and scale well in practice due to advancements in heuristics developed for them. New heuristics are introduced often, through research publications or as improvements to solvers participating in annual SAT/SMT competitions [16, 67].

Decision (branching) heuristics are used to decide free variables and their values and thus are crucial for the scalability of SAT solving by reducing assignment mistakes. We will adapt well-known greedy heuristics [67] such as VSIDS [83] (Variable State Independent Decaying Sum), which prioritizes variables not appeared in recently learned clauses, and DLIS [24] (Dynamic Largest Individual Sum), which chooses unassigned literal that satisfies the maximum number of unsatisfied clauses. We will also explore other strategies such as *random restart* and *clause deletion* to avoid being stuck due to greedy choices and improve memory usage [83].

Moreover, instead of searching over activation variables that abstract neuron values, we will explore searching over variables that abstract input ranges (e.g., splitting the range of each input into k regions as suggested in [85, 109, 139]). For instance, the DNN in Fig. 1 we can use two boolean variables v_1, v_2 to abstract the ranges of inputs x_1, x_2, respectively, e.g., for $x_1 \in [-1, 1]$, $v_1 = T$ means $-1 \leq x_1 < 0$ and $v_1 = F$ means $0 \leq x_1 \leq 1$. This would make the complexity of NeuralSAT depend on the *input dimensions*, instead of the size (# of neurons), of the networks. Thus, when combined with CDCL and abstraction in NeuralSAT, this strategy can improve scalability as real-world DNNs often have more neurons than their inputs [109] (e.g., AlexNet used for image

Again, reusing the example in Fig. 1 to demonstrate the proposed idea.

classification has nearly 6.5×10^6 neurons while training images have far fewer dimensions, e.g., 28x28 for MNIST [71] and 32x32x3 for CIFAR [65] benchmarks). NeuralSAT will have both search algorithms and switch them depending on the input DNNs.

The theory solver of NeuralSAT currently uses the common "forward" approach to compute and propagate abstraction of the DNN starting with the input layers to the output layers and checks if the approximated result covers the property being considered. We will explore the opposite *"backward"* propagation from program analysis [10], which starts from the considered property at the output layer and propagates computation back to the input layer, and checks if the result satisfies the input conditions of the DNN. The directed symbolic execution work in [75] presents various strategies to guide computation flow to the considered property that might be applicable to our approximation. These approaches involve the property of interest earlier and thus help improve performance [75].

4.2.3 Engineering improvements matter!

Consider nnenum [7, 9], which applies a set of six optimizations to the existing path enumeration-based technique used in the `NNV` tool [131–134]. These engineering improvements are often not considered scientific contributions (e.g., compared to new abstract domains or search algorithms). Yet, we cannot deny the effectiveness of these optimizations, which allow nnenum to perform 120x faster [9] than `NNV` and become the fastest tool for FNNs in VNN-COMP'21 while NNV ranks 10th.

Currently, our NeuralSAT prototype is unoptimized, making it amenable to many possible optimizations. For example, to *leverage multi-thread processing*, we will try the Split and Conquer technique [61, 145], used to parallelize Marabou by partitioning input regions and applying the search to each smaller region individually. We will also explore the Branch and Bound (BaB) technique [9, 19, 138, 140], which is adopted in many top DNN verifiers (e.g., nnenum[2], β-CROWN) and splits the search at ReLU (one search on the path when the neuron is active and another on the path when it is inactive). We will also adopt the increasingly popular optimization of lifting heavy matrix computations (e.g., during abstraction in the theory solver) to *GPUs* [85, 125, 138, 140]. We also plan to rewrite NeuralSAT, currently implemented in Python, using a *compiled language* such as C or Rust, and have APIs in Python, Java, and OCaml. This API practice is used by the CVC [12] and Z3 [26] SMT solvers and contributes to their widespread adoption.

Using a quote from a well-known researcher to show the need for engineering, which is what this subtask is about.

Ken McMillan, a prominent researcher in model checking and formal methods, once noted that *"Engineering matters: you can't properly evaluate a technique without an efficient implementation"* [81]. Indeed, an optimized tool would allow us to evaluate the NeuralSAT approach more accurately and find bottlenecks that are worth improving.

4.3 RC#2: Verifying and Finding Bugs in NeuralSAT

High level description of the RC. Listing the two questions the RC tackles helps mix things up a bit to avoid boring writing. Note that I also show some concrete evidence of the problem obtained from my experimentation.

Just as with other programs, DNN verification tools can have bugs [50]. Our experience with Marabou, shown in Tab. 1, finds that its results can be incorrect[3]. Given that formal verification tools are often employed for mission-critical tasks, it is important that their results could be trusted, especially when they claim that the networks are safe or robust to adversarial attacks.

This RC tackles two questions: (i) how do we verify the results of the verifiers? and (ii) how do we find unsound bugs in the verifiers? For the first question, we will extract *resolution proofs* and *unsat core* to verify the results of NeuralSAT. For the second question, we will explore the highly-successful *metamorphic testing* techniques [69, 103, 126, 143, 144] that find bugs in mature compilers

[2] [redacted], will help with optimization ideas (LoC attached).

[3] E.g., in the ACAS XU experiment in Tab. 1 Marabou proved invalid properties (e.g., ϕ_2 for 2 networks `model_2_9,model_3_6`) and disproved valid ones (e.g., ϕ_1 for 3 networks `model_2_8, model_5_5, model_5_8`).

and SMT solvers. The research here will allow us to find unsound bugs in NeuralSAT and other DNN verifiers during production and certify their results during deployment.

4.3.1 Certifying Proved Results

When the verifier disproves a property (e.g., returning sat in NeuralSAT), we can run the DNN on the counterexample and confirm that it violates the property. However, when the verifier proves a property (returning unsat and no counterexamples), how can we certify such a claim?

An effective solution for this question would depend on the algorithm used by the verifier. For example, a recent work [50] adapts Farkas's lemma to show unsatisfiability proof of linear constraints used in Marabou. Our insight is that CDCL already includes the mechanism to produce unsat proofs, which we can extend with minimal overhead to extract proof certificates.

Resolution Proofs. One way to verify unsatisfiability result is to check the steps taken by the verifier that result in unsat. For NeuralSAT, if a formula is unsatisfiable, we will at some point learn a new clause that is inconsistent with existing ones (at which point NeuralSAT returns unsat). For instance, in the fifth iteration of the example in §4.1.1, NeuralSAT learns the clause $\overline{v_3} \vee \overline{v_4}$, which is inconsistent with two learned clauses $\{v_3, \overline{v_3} \vee v_4\}$, and returns unsat. Thus, we will exploit clause learning in NeuralSAT to produce unsatisfiability proofs.

Reusing the example in §4.1.1 to illustrate the proposed idea.

More specifically, NeuralSAT already uses an *implication graph* to store reasons about clauses it learns and to infer new assignments. Using ideas from SAT solving [5, 67, 155], we can use the implication graph to build a *resolution graph*, which tracks the order and reasons clauses were learned. In the end, when we learn a clause inconsistent with other clauses and results in unsat, we unwind the resolution graph to extract steps from the last clause to the original clauses, i.e., a *resolution proof* showing unsat through learned clauses. Moreover, we will apply techniques in SAT solving to *reduce* resolution proofs, making them smaller and easier to check [154, 155].

Unsat Core. Another way to show unsatisfiability is using an *unsat core*, which is an unsatisfiable subset of the original set of clauses. In NeuralSAT, each clause asserts that a hidden neuron is either active or inactive, thus an unsat core means an unsatisfiable activation pattern.

To obtain unsat cores, we will reuse resolution graphs, e.g., by traversing the resolution graph to capture all original clauses—this represents an unsat core that was used to derive unsatisfiability. We will further minimize the obtained unsat core using the fixed-point iterative technique described in [154]. The work in [41] surveys well-known approaches, many of which exploit resolution graphs and proofs [27, 33, 111], to extract small unsat cores.

Unsat cores are also useful for other debugging e.g., the well-known Alloy specification analyzer [53] and our fault localization work [159] use unsat cores to identify inconsistencies in a specification. Similarly, a user of NeuralSAT can use unsat cores to understand why an unexpected property is valid in a network or why a network cannot produce a counterexample for a presumably invalid property.

4.3.2 Testing Verifiers

In §4.3.1 we propose techniques to certify the results of NeuralSAT to gain users' trust, e.g., during deployment. Here, we will develop stress-testing techniques to generate inputs that trigger soundness bugs in NeuralSAT and other DNN verifiers before deployment. This problem of testing DNN verifiers is both important (to verify DNNs deployed in safety-critical domains) and challenging (e.g., mature verification tools are likely well-tested).

To tackle this problem, we will explore the *semantic fusion* technique [144] that generates SMT input formulae to test SMT solvers. The main idea is to "fuse" pairs of either sat (or unsat) formulae

by concatenating and substituting variables in the concatenated formula with inverted expressions to produce another sat (or unsat) formulae—if the SMT solver returns an incorrect result on the fused formula then it has a bug. This technique, implemented in the Yin Yang tool [104], has found thousands of confirmed bugs in the mature CVC and Z3 SMT solvers.

We will apply semantic fusion to DNN verification by applying Yin Yang as a blackbox to SMT formulae encoding DNN verification. Specifically, we take two satisfiable DNN verification instances, convert them into two SMT formulae, and apply Yin Yang to create a fused satisfiable SMT formula, from which we create a DNN verification instance and feed it to NeuralSAT or other DNN verifiers to check for bugs (e.g., if NeuralSAT returns unsat). This blackbox approach also allows us to use other effective SMT-based fuzzing tools (e.g., FuzzSMT [18] and Storm [76]).

Moreover, we will explore the *EMI* (equivalence modulo input) compiler-testing techniques [69, 126] by running a seed DNN on sample inputs, analyzing its path, and fuzzing the DNN based on the traces (e.g., deleting unused neurons and connection edges). EMI-based techniques and tools have successfully found thousands of confirmed bugs in the GCC and LLVM compilers, and the benefit of these methods is that they require one input instance instead of two as in semantic fusion.

4.4 RC#3: Graph Neural Networks

High level description and motivation. This section on GNNs is a bit separate from the other proposed work and received mixed reviews. Some reviewers found that it is a good exploration while others thought it doesn't connect to other RCs.

Graph Neural Networks (GNNs) is another powerful model of deep learning and have been applied to many practical problems, e.g., heuristics for NP-Complete problems [15, 64, 117, 118], knowledge graphs analysis [136], recommendation systems for social networks [150], chemical and protein classification [34, 105], and advanced COVID-19 detection [112, 162] and vaccine development [21, 47, 161]. However, while many formal techniques and tools have been developed for FNNs, few exist for GNNs, potentially due to the dynamic structure and behavior of GNNs (e.g., they take arbitrary input graphs and change computations based on the structures of the graphs).

This RC aims to analyze GNNs by reducing them into FNNs, allowing for the transfer and applications of techniques developed for FNNs to GNNs. Key challenges are that creating a single FNN that accepts all possible input graphs of a GNN or an FNN for each input graph that a GNN accepts would likely be infeasible and not scale. Our insight is to extract substructures of input graphs that influence the GNN's prediction and use them to create FNNs. Our pilot study shows the potential and an interesting application of the approach: we were able to discover properties of GNNs using an invariant generation tool for FNNs.

4.4.1 Reducing GNNs to FNNs

Instead of creating a GNN for each input graph, we create FNNs for classes of inputs using influential substructures. *Influential substructures* of the input graphs are compact subgraphs and subsets of nodes of those graphs that have a crucial role in GNN's prediction. For example, inputs that a GNN predicts "like Johnny Depp" might have substructures with strong connections to nodes representing interests in "pirates" or recent searches for "defamation".

To obtain influential substructures, we will *automatically infer* them from sample data (e.g., the training data of the GNN). We will leverage existing GNN techniques and tools such as GNNExplainer [151] and PGExplainer [74] to mine influential substructures from sample graphs. For example, in Fig. 4 GNNExplainer extracts three substructures $G_{sub1}, G_{sub2}, G_{sub3}$ that are common from the input graphs G_1, G_2 and contribute significantly to the prediction of the red target node.

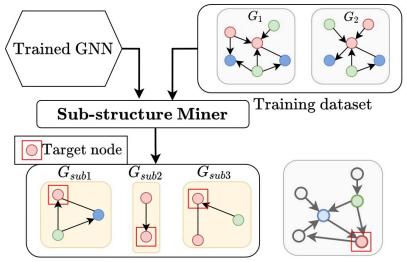


Fig. 4: Influential substructures and graph isomorphism.

Example can be effective and makes technical stuff less boring.

We also need to check if a substructure covers an input graph (e.g., to evaluate and improve the accuracy of mined substructures or to reject (bad) input graphs). To do this, we will check if the graph and the substructure, which is also a graph, are *isomorphic*, e.g., in Fig. 4 the lower-right graph and substructure G_{sub1} are isomorphic. Graph isomorphism is well-known, and we can check it using existing techniques [17, 42] and tools (e.g., nauty and Traces [80], bliss [57], conauto [73]).

Finally, once having influential substructures, we create an FNN by *unrolling* the message passing operations [34] of the GNN over the structures. More specifically, the FNN will have neurons and connections representing the computation of message update operations of the GNN (e.g., updating a value of a node in the graph based on the information of its neighboring nodes). Then we combine created FNNs to represent the behavior of the GNN.

4.4.2 Pilot Study

The published short ICSE NIER paper [98] and this pilot study help demonstrate the feasibility of the approach. Having concrete evidence such as prelim work or pilot study greatly strengthen the proposal.

We have introduced this idea of reducing GNNs to FNNs using substructures in a recent ICSE'22 NIER paper [98]; this RC will develop it. For this proposal we conducted a pilot study to assess the approach using a GNN modeling a simple BFS algorithm, which predicts if a node n in a given graph would be visited next [137]. We created a small GNN and trained it using existing data from the work in [137] (e.g., the GNN for BFS has only 2 layers and its training data has 160 input graphs). Then, we manually performed the proposed tasks on the GNN. First, we used GNNExplainer to extract substructures from the training data used for that GNN. Then, we apply scripts to perform unrolling to obtain FNNs for each substructure and manually combine them into an FNN.

To evaluate the application of the approach, we applied the invariant discovery tool Prophecy [37] to the reduced FNN to learn its properties. While Prophecy produces cryptic formulae representing discovered properties, we recognized several familiar and crucial ones such as (i) if n has been visited, it will not be visited next regardless of the state of its neighbors, (ii) if n and all of its neighbors have not been visited, then n would not be visited next, and (iii) if all neighbors of n have been visited, then n will be visited next. We also were able to use Marabou (used by Prophecy to check inferred properties) to confirm the discovered properties are valid in the reduced FNN.

While we anticipate scalability (e.g., the size of the created FNN might be big) and other challenges (e.g., how to interpret discovered invariants), we are excited about these challenges and opportunities they would provide (e.g., GNNExplainer and Prophecy show that substructures and DNN invariants are better understood by humans through visualization, i.e., useful for explainable AI [37, 115, 151]). Similar to program analyses being benefited through advancements in constraint solving, this research direction will become more realizable as DNN analyses and tools (such as those developed in this project) become more available and powerful.

4.5 Evaluation

Evaluation or Validation Plan is especially important in this type of topics of prototype and tool developments. In general, every proposal should have this part to evaluate the outcomes of the project.

Benchmarks We will continue using VNN-COMP benchmarks, which include standard ones such as ACAS XU [56], MNIST [71], CIFAR [65], and others designed to challenge state-of-the-art verifiers. Together, these benchmarks consist of thousands of instances for FNNs, CNNs, and ResNet [8]. We also use more diverse benchmarks including those automatically generated [146]. For RNNs we will use benchmarks from verification and invariant generation work, e.g., [54, 82]. For GNNs we will collect those modeling classical algorithms such as sorting [107] and shortest paths [39, 137, 147] (e.g., BFS, Bellman-Ford, and Prim's)—these have well-known properties that we can use as ground truths (e.g., to compare inferred invariants). We will also use GNNs modeling complicated tasks, e.g., heuristics for the Traveling Salesman [15, 64] and boolean satisfiability problems [117, 118], and large real-world analyses, e.g., the MUTAG (mutagenic effects of molecules) and REDDIT-BINARY

(social network analysis) benchmarks used in GNNExplainer for substructure mining [151].

Evaluation Metrics For RCs#0–1 we will evaluate and compare the performance of NeuralSAT (e.g., correctness, efficiency, memory usage) to state-of-the-art tools such as those participating in VNN-COMPs. We also compare to newer versions of published tools as DNN verification evolve rapidly and developers often publish new tools to their code repositories. For RC#1, we will compare the precision and scalability of weak max-plus to existing abstract domains, and evaluate improvements of individual and combinations of heuristics and optimizations using the current unoptimized NeuralSAT prototype as the baseline for comparison. We will participate in VNN-COMPs [8], which systematically compare tools over a wide variety of benchmarks and criteria.

For RC#2 we will evaluate the overhead of computing resolution proofs, and report the sizes of unsat cores and resolution proofs. We will work with the organizers of VNN-COMPs to give bonus points for verifiers to include proofs of proved results and adopt a standard format for such proofs (similar to the DRAT [31,142] format used in SAT competitions for unsatisfiability proofs). We will evaluate stress-testing on NeuralSAT and other mature and well-maintained DNN verifiers (because they would likely have fewer bugs). We will record efficiency and effectiveness (e.g., number of bugs found), and report bugs to DNN tool developers. The Yin Yang tool has helped fixed thousands of real bugs in Z3 and CVC [104]; we believe our research will be as effective, if not more, for DNN verifiers, which are not as mature and well-tested compared to these SMT solvers.

For RC#3 we focus on precision (able to obtain relevant substructures for reduction), efficiency (in mining substructures and performing the reduction), scalability (sizes of created FNNs), and applications (whether existing FNN techniques such as NeuralSAT and Prophecy work on the reduced FNNs). This research also produces new, diverse benchmarks consisting of GNNs' modeling tasks that are often not used by FNNs. We will share these benchmarks and our results with FNN tool developers to improve their work.

4.6 Related Work

Not sure if this section is needed, e.g., I have other funded proposals without this sect. But for this CAREER it seems necessary to demonstrate that I know the literature, especially since I am new in this topic and to show the relationship/difference of the RCs with related work.

DNN Verification The literature on DNN verification is rich (cf. [72, 135]), and here we summarize well-known techniques with tool implementations. Constraint-based approaches such as DLV [49], Planet [28], Reluplex [59], Marabou [61] (successor of Reluplex) transform DNN verification into a constraint problem, solvable using an SMT (Planet, DLV) or customized simplex and MILP (Reluplex, Marabou) solvers. Abstraction-based approaches such as AI^2 [32], ERAN [85,123,125] (DeepZ [123], DeepPoly [125], K-ReLU [122]), RefineZono [124]), ReluVal [139], Neurify [138], VeriNet [46], NNV [133], nnenum [7, 9], CROWN [153], β-CROWN [140], use abstract domains such as interval (ReluVal/Neurify), zonotope (DeepZ, nnenum), polytope (DeepPoly), starset/imagestar (NNV, nnenum) to scale verification. OVAL [101] and DNNV [119] are frameworks containing various existing techniques and tools. Our NeuralSAT is a constraint-based approach that integrates clause learning and abstractions.

Abstractions, Heuristics, and Optimizations The aforementioned interval, zonotope, polytope, and starset/imagestar are well-known and effective abstraction domains. Some verifiers such as ERAN and nnenum use multiple abstractions (e.g., ERAN uses zonotope and polytope, nnenum adopts zonotope and imagestar). The work in [38] uses the general max-plus abstraction [45] to represent the non-convex behavior of ReLU. NeuralSAT currently uses polytope in its theory solver and RC#1 will develop the "weak max-plus" abstraction, which is more restrictive, but likely much faster than general max-plus.

Modern SAT solving benefits from effective heuristics, e.g., VSIDS and DLIS strategies for

decision (branching), and random restart [83] and shortening [22] or deleting clauses [83] for memory efficiency and avoiding local maxima caused by greedy strategies (§4.2.2). Similarly, modern DNN verifiers such as nnenum, ERAN, β-CROWN, and Marabou include many optimizations to improve performance, e.g., Branch and Bound [19] and Split and Conquer [61, 145] for parallelization, and various optimizations for abstraction refinement [7, 124]) and bound tightening [7, 61, 140]. RC#1 will explore such heuristics and optimizations to improve the performance of NeuralSAT.

Certifying and Testing DNN Verifiers Certifying proved results are well-studied and even required in SAT competitions [13, 31]. In DNN verification, the recent work [50] constructs unsat proofs for Marabou by combining Farkas's lemma to show the unsatsifiability of linear constraints with a mechanism to handle ReLU constraints. For stress-testing, while numerical imprecision in DNN verifiers is well-acknowledged (e.g., [7, 55]), we are unaware of testing techniques making the verifiers reverse their decisions (e.g., from sat to unsat and vice versa). In contrast, metamorphic testing [20] (e.g., semantic fusion [104, 144] and EMI [69, 126]) and fuzzing/mutation approaches (e.g., [18, 76, 103, 143]) have been used to find many bugs in mature SMT solvers and compilers. RC#2 will develop resolution proofs for proved results and apply SMT testing-based techniques to find bugs in DNN verifiers.

This proposal does not include a timeline diagram and section, which many proposals have. I had created it but felt it doesn't give anything useful, i.e., mostly bs, and decided to not include it.

Reductions and GNNs Reduction techniques have been explored in DNN analyses, e.g., converting DNN correctness properties into adversarial attacks that can be handled by existing adversarial techniques [120], and reducing RNNs into FNNs through loop unrolling [1] or approximation [54]. In general, reduction is commonly used to solve unknown problems, and is thus a good approach to tackle GNNs, in which few formal verification techniques or tools exist. However, while few verifiers exist for GNNs, work such as GNNGuard [157] can help defend against adversarial attacks and data poisoning [163–165] in GNNs by augmenting the networks and retraining them to improve robustness. Such defense mechanisms complement the research in RC#3, which aims to apply FNN verification and property discovery analyses to GNNs.

5 Integration of Research and Education

The proposed research and funding will help me continue my current efforts and explore new activities in integrating research into my teaching and involving undergraduate and minority students in my research. Planned activities include new course development, an interactive "DNN Verification by examples" book, and efforts to involve K-12 students.

Crucial section in the CAREER that *intergrate* research and education. Think hard, be creative, and customize it based on your *own* experience. Note that I don't strictly use "we/our" here in this part. Another note is the section has just a little bit more than a page, which is very short compared to most CAREER proposals. But I felt (and reviewers probably concurred) that it is creative and fits my experience.

5.1 Curriculum Development and "DNN Verification by Examples" Book

At GMU, I have redesigned my graduate course, Software Construction and Specification, to include verification and constraint solving (using Z3). These topics receive positive feedback from students, many of whom are professional developers and unfamiliar with formal methods. Due to high demand, I am designing an online undergraduate version of the course with Wiley publishing. This version, available in Fall'23, will include DNNs as a practical application and integrate techniques and tools developed in this research. Over the time frame of this proposal, I will teach the course (avg. 40 students) at least five times and thus have ample opportunities to evaluate and improve its materials.

DNN Book Learning DNN verification by reading technical papers is hard. My students and I have started a "survey" paper reviewing major DNN analyses. In contrast to existing DNN surveys and books, e.g., [2, 72, 135], this paper demonstrates how major DNN verification techniques work step-by-step using small DNN examples (e.g., similar to the DNN in Fig. 1 used in various examples in this proposal). I will extend this activity into an online, interactive, and open-source book, similar

Reviewers like this idea of creating a Jupter notebook demonstrating DNN verification techniques.

to the Fuzzing Book [152] used in my previous Software Testing courses. This book would contain small executable Jupyter implementations of DNN techniques, allowing students to experiment with code and their effects live on the web browser.

I will integrate materials from the book into my Software Construction and Specification course and evaluate its materials along with the course. Many of my collaborators teach similar software courses and thus are potential users of the book (e.g., [redacted] has expressed interest to contribute and use particular modules of the book in [redacted] courses).

5.2 Undergraduate Research and Outreach

Talk about experience and success of working with undregrads to convince the reader that you are capable of proposed outreach activity.

Undergraduate Research I enjoy involving and mentoring undergraduate students in my research and motivating them to pursue graduate studies. Since 2017, I have worked with 10 undergraduate research assistants (most of whom are first-generation students, two females, and two freshmen). One former undergraduate student, Linhan, has recently moved to GMU and started his Ph.D. with me. Another student, KimHao, has worked with me since his freshman year, produced numerous publications [51, 52, 86, 87, 93–96], and is planning to pursue a Ph.D. at GMU working on this DNN project. I will continue recruiting and creating opportunities for undergraduate students and have included a budget to support an undergraduate researcher in the proposal (I will also apply for an REU supplement if the project is funded).

Presenting concrete evidence showing my outreach experience.

Community Outreach At UNL (University of Nebraska-Lincoln), my previous institution, I was involved in Initialize, a program developed by CS undergraduates dedicated to using computing skills to give back to the local community. In particular, in 2019 I worked with [redacted] and Initialize students and obtained funding ($19K from Bosch Community Fund and $5K from Beyond School Bell, a local nonprofit organization) for several new initiatives (e.g., providing CS learning opportunities to under-resourced and underrepresented students in Lincoln).

Outreach plan tailored for my situation

I will build on these experiences and continue similar outreach activities at GMU, where I just moved to earlier this year. For example, I live in the southern part of Prince William County (PWC), about an hour from GMU. It currently does not have connections with GMU, and has a much more diverse and low-income population (compared to other places in Northern Virginia according to the US Census). I aim to build a bridge between GMU and PWC K-12 schools and have already begun coordination with [redacted] at [redacted], where my children attend. He is interested in doing activities with GMU, and I am working with several CS faculty at GMU to get several events going [redacted] and GMU faculty presenting their work to 4th and 5th graders at [redacted] (Asst. Profs. [redacted] and [redacted] have expressed interest). Moreover, using my experience in proposal writing, I plan to work with them to obtain funding from PWC, e.g., materials to build/support a stem lab/maker space. I will evaluate these efforts via the number of events and funding created and levels of student and teacher involvement.

6 Results from Prior NSF Support

Mention potentional relationship between the CAREER and a prior NSF project.

Nguyen is the PI on CCF-2107035 *"Ensuring Safety and Liveness of Modern Systems through Dynamic Temporal Analysis"* ($399,879, 2021–2024). **Intellectual Merit:** This project develops theoretical and practical integration of static and dynamic approaches to analyze and repair temporal aspects of reactive/interactive systems. NeuralSAT might be extended to prove dynamically inferred temporal properties of RNNs. **Broader Impacts:** This work produces advanced methods for ensuring the safety and liveness of today's reactive/interactive software. The PI is working with a graduate and undergraduate on this project and has produced two publications [52, 95].

References

I spent sometime to clean up these references, making them consistent, and ensuring that nothing "bad" stands out.

[1] M. E. Akintunde, A. Kevorchian, A. Lomuscio, and E. Pirovano. Verification of rnn-based neural agent-environment systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6006–6013, 2019.

[2] A. Albarghouthi et al. Introduction to neural network verification. *Foundations and Trends® in Programming Languages*, 7(1–2):1–157, 2021.

[3] X. Allamigeon, S. Gaubert, and E. Goubault. Inferring min and max invariants using max-plus polyhedra. In *International Static Analysis Symposium*, pages 189–204. Springer, 2008.

[4] X. Allamigeon, S. Gaubert, and E. Goubault. The tropical double description method. In *27th International Symposium on Theoretical Aspects of Computer Science-STACS 2010*, pages 47–58, 2010.

[5] R. Asín, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell. Efficient generation of unsatisfiability proofs and cores in sat. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, pages 16–30. Springer, 2008.

[6] J. Bai, F. Lu, K. Zhang, et al. Onnx: Open neural network exchange. `https://onnx.ai`, last accessed July 8, 2024.

[7] S. Bak. nnenum: Verification of relu neural networks with optimized abstraction refinement. In *NASA Formal Methods Symposium*, pages 19–36. Springer, 2021.

[8] S. Bak, C. Liu, and T. Johnson. The Second International verification of Neural Networks Competition (VNN-COMP 2021): Summary and Results. *arXiv preprint arXiv:2109.00498*, 2021.

[9] S. Bak, H.-D. Tran, K. Hobbs, and T. T. Johnson. Improved geometric path enumeration for verifying relu neural networks. In *International Conference on Computer Aided Verification*, pages 66–96. Springer, 2020.

[10] R. Baldoni, E. Coppa, D. C. D'elia, C. Demetrescu, and I. Finocchi. A survey of symbolic execution techniques. *ACM Computing Surveys (CSUR)*, 51(3):1–39, 2018.

[11] M. Balunovic, M. Baader, G. Singh, T. Gehr, and M. Vechev. Certifying geometric robustness of neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[12] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli. Cvc4. In *International Conference on Computer Aided Verification*, pages 171–177. Springer, 2011.

[13] C. Barrett, L. De Moura, and P. Fontaine. Proofs in satisfiability modulo theories. *All about proofs, Proofs for all*, 55(1):23–44, 2015.

[14] R. J. Bayardo Jr and R. Schrag. Using csp look-back techniques to solve real-world sat instances. In *Aaai/iaai*, pages 203–208. Providence, RI, 1997.

[15] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio. Neural combinatorial optimization with reinforcement learning. *ICLR Workshop*, 2017.

[16] D. Beyer. Progress on software verification: Sv-comp 2022. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 375–402. Springer, 2022.

[17] F. Bi, L. Chang, X. Lin, L. Qin, and W. Zhang. Efficient subgraph matching by postponing Cartesian products. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 26-June-20:1199–1214, 2016.

[18] R. Brummayer and A. Biere. Fuzzing and delta-debugging smt solvers. In *Proceedings of the 7th International Workshop on Satisfiability Modulo Theories*, pages 1–5, 2009.

[19] R. Bunel, P. Mudigonda, I. Turkaslan, P. Torr, J. Lu, and P. Kohli. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21(2020), 2020.

[20] T. Y. Chen, S. C. Cheung, and S. M. Yiu. Metamorphic testing: a new approach for generating next test cases. *arXiv preprint arXiv:2002.12543*, 2020.

[21] M. Cheung and J. M. Moura. Graph neural networks for covid-19 drug discovery. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 5646–5648. IEEE, 2020.

[22] J. W. Chinneck and E. W. Dravnieks. Locating minimal infeasible constraint sets in linear programs. *ORSA Journal on Computing*, 3(2):157–168, 1991.

[23] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, 1971.

[24] F. Copty, L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella, and M. Y. Vardi. Benefits of bounded model checking at an industrial setting. In *International Conference on Computer Aided Verification*, pages 436–453. Springer, 2001.

[25] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[26] L. De Moura and N. Bjørner. Z3: An efficient SMT Solver. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4963 LNCS:337–340, 2008.

[27] N. Dershowitz, Z. Hanna, and A. Nadel. A scalable algorithm for minimal unsatisfiable core extraction. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 36–41. Springer, 2006.

[28] R. Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pages 269–286. Springer, 2017.

[29] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1625–1634, 2018.

[30] S. Forrest, T. Nguyen, W. Weimer, and C. Le Goues. A genetic programming approach to automated software repair. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 947–954, 2009.

[31] N. Froleyks, M. Heule, M. Iser, M. Järvisalo, and M. Suda. Sat competition 2020. *Artificial Intelligence*, 301:103572, 2021.

[32] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2018.

[33] R. Gershman, M. Koifman, and O. Strichman. Deriving small unsatisfiable cores with dominators. In *International Conference on Computer Aided Verification*, pages 109–122. Springer, 2006.

[34] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural Message Passing for Quantum Chemistry. 4 2017.

[35] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `https://www.deeplearningbook.org`, last accessed July 8, 2024.

[36] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[37] D. Gopinath, H. Converse, C. Pasareanu, and A. Taly. Property inference for deep neural networks. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 797–809. IEEE, 2019.

[38] E. Goubault, S. Palumby, S. Putot, L. Rustenholz, and S. Sankaranarayanan. Static analysis of relu neural networks with tropical polyhedra. In *International Static Analysis Symposium*, pages 166–190. Springer, 2021.

[39] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

[40] C. Grebner, H. Matter, D. Kofink, J. Wenzel, F. Schmidt, and G. Hessler. Application of deep neural network models in drug discovery programs. *ChemMedChem*, 16(24):3772–3786, 2021.

[41] A. Griggio. *An effective SMT engine for Formal Verification*. PhD thesis, University of Trento, 2009.

[42] M. Grohe and P. Schweitzer. The graph isomorphism problem. *Communications of the ACM*, 63(11):128–134, 2020.

[43] V. Gulshan, L. Peng, M. Coram, M. C. Stumpe, D. Wu, A. Narayanaswamy, S. Venugopalan, K. Widner, T. Madams, J. Cuadros, et al. Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *Jama*, 316(22):2402–2410, 2016.

[44] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[45] B. Heidergott, G. J. Olsder, J. Van Der Woude, and J. van der Woude. *Max Plus at work: modeling and analysis of synchronized systems: a course on Max-Plus algebra and its applications*, volume 13. Princeton University Press, 2006.

[46] P. Henriksen and A. Lomuscio. Efficient neural network verification via adaptive refinement and adversarial search. In *ECAI 2020*, pages 2513–2520. IOS Press, 2020.

[47] K.-L. Hsieh, Y. Wang, L. Chen, Z. Zhao, S. Savitz, X. Jiang, J. Tang, and Y. Kim. Drug repurposing for covid-19 using graph neural network with genetic, mechanistic, and epidemiological validation. *Research Square*, 2020.

[48] X. Huang, D. Kroening, W. Ruan, J. Sharp, Y. Sun, E. Thamo, M. Wu, and X. Yi. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Computer Science Review*, 37:100270, 2020.

[49] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. Safety verification of deep neural networks. In *International conference on computer aided verification*, pages 3–29. Springer, 2017.

[50] O. Isac, C. Barrett, M. Zhang, and G. Katz. Neural network verification with proof production. *Proc. 22nd Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, 2022.

[51] D. Ishimwe, K. Nguyen, and T. Nguyen. Dynaplex: analyzing program complexity using dynamically inferred recurrence relations. *Proc. ACM Program. Lang.*, 5(OOPSLA):1–23, 2021.

[52] D. Ishimwe, T. Nguyen, and K. Nguyen. Dynaplex: Inferring asymptotic runtime complexity of recursive programs. In *2022 IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 61–64. IEEE, 2022.

[53] D. Jackson, I. Schechter, and H. Shlyahter. Alcoa: the alloy constraint analyzer. In *Proceedings of the 22nd international conference on Software engineering*, pages 730–733, 2000.

[54] Y. Jacoby, C. Barrett, and G. Katz. Verifying recurrent neural networks using invariant inference. In *International Symposium on Automated Technology for Verification and Analysis*, pages 57–74. Springer, 2020.

[55] K. Jia and M. Rinard. Exploiting verified neural networks via floating point numerical error. In *International Static Analysis Symposium*, pages 191–205. Springer, 2021.

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

[57] T. Junttila and P. Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In D. Applegate, G. S. Brodal, D. Panario, and R. Sedgewick, editors, *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments and the Fourth Workshop on Analytic Algorithms and Combinatorics*, pages 135–149. SIAM, 2007.

[58] D. Kapur, Z. Zhang, M. Horbach, H. Zhao, Q. Lu, and T. Nguyen. Geometric quantifier elimination heuristics for automatically generating octagonal and max-plus invariants. In *Automated Reasoning and Mathematics*, pages 189–228. Springer, 2013.

[59] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[60] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Towards proving the adversarial robustness of deep neural networks. *Proc. 1st Workshop on Formal Verification of Autonomous Vehicles (FVAV), pp. 19-26*, 2017.

[61] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, et al. The marabou framework for verification and analysis of deep neural networks. In *International Conference on Computer Aided Verification*, pages 443–452. Springer, 2019.

[62] A. I. Khan, J. L. Shah, and M. M. Bhat. Coronet: A deep neural network for detection and diagnosis of covid-19 from chest x-ray images. *Computer methods and programs in biomedicine*, 196:105581, 2020.

[63] M. J. Kochenderfer, J. E. Holland, and J. P. Chryssanthacopoulos. Next-generation airborne collision avoidance system. Technical report, Massachusetts Institute of Technology-Lincoln Laboratory Lexington United States, 2012.

[64] W. Kool, H. van Hoof, and M. Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2018.

[65] A. Krizhevsky, V. Nair, and G. Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55(5), 2014.

[66] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[67] D. Kroening and O. Strichman. *Decision procedures*. Springer, 2016.

[68] T. C. Le, T. Antonopoulos, P. Fathololumi, E. Koskinen, and T. Nguyen. Dynamite: dynamic termination and non-termination proofs. *Proceedings of the ACM on Programming Languages*, 4(OOPSLA):1–30, 2020.

[69] V. Le, M. Afshari, and Z. Su. Compiler validation via equivalence modulo inputs. *ACM Sigplan Notices*, 49(6):216–226, 2014.

[70] C. Le Goues, T. Nguyen, S. Forrest, and W. Weimer. Genprog: A generic method for automatic software repair. *Ieee transactions on software engineering*, 38(1):54–72, 2011.

[71] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010.

[72] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barrett, M. J. Kochenderfer, et al. Algorithms for verifying deep neural networks. *Foundations and Trends® in Optimization*, 4(3-4):244–404, 2021.

[73] J. L. López-Presa, L. N. Chiroque, and A. Fernández Anta. Novel techniques for automorphism group computation. In *International Symposium on Experimental Algorithms*, pages 296–307. Springer, 2013.

[74] D. Luo, W. Cheng, D. Xu, W. Yu, B. Zong, H. Chen, and X. Zhang. Parameterized explainer for graph neural network. *Advances in Neural Information Processing Systems*, 33, 2020.

[75] K.-K. Ma, K. Yit Phang, J. S. Foster, and M. Hicks. Directed symbolic execution. In *International Static Analysis Symposium*, pages 95–111. Springer, 2011.

[76] M. N. Mansur, M. Christakis, V. Wüstholz, and F. Zhang. Detecting critical bugs in smt solvers using blackbox mutational fuzzing. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 701–712, 2020.

[77] B. Mariano, J. Reese, S. Xu, T. Nguyen, X. Qiu, J. S. Foster, and A. Solar-Lezama. Program synthesis with algebraic library specifications. *Proceedings of the ACM on Programming Languages*, 3(OOPSLA):1–25, 2019.

[78] J. Marques Silva and K. Sakallah. Grasp-a new search algorithm for satisfiability. In *Proceedings of International Conference on Computer Aided Design*, pages 220–227, 1996.

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

[80] B. D. McKay and A. Piperno. Practical graph isomorphism, ii. *Journal of symbolic computation*, 60:94–112, 2014.

[81] K. McMillan. A perspective on formal verification. In *David Dill @ 60 Workshop, colocated with CAV*, 2017.

[82] S. Mohammadinejad, B. Paulsen, J. V. Deshmukh, and C. Wang. DiffRNN: Differential Verification of Recurrent Neural Networks. pages 117–134, 2021.

[83] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535, 2001.

[84] L. d. Moura and N. Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[85] C. Müller, F. Serre, G. Singh, M. Püschel, and M. Vechev. Scaling polyhedral neural network verification on gpus. *Proceedings of Machine Learning and Systems*, 3:733–746, 2021.

[86] K. Nguyen and T. Nguyen. Gentree: Using decision trees to learn interactions for configurable software. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1598–1609. IEEE, 2021.

[87] K. Nguyen, T. Nguyen, and Q.-S. Phan. Analyzing the cmake build system. In *2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 27–28. IEEE, 2022.

[88] T. Nguyen. The DIG Invariant Generation System. `https://github.com/dynaroars/dig/`, last accessed July 8, 2024.

[89] T. Nguyen, M. B. Dwyer, and W. Visser. Symlnfer: Inferring program invariants using symbolic states. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 804–814. IEEE, 2017.

[90] T. Nguyen, D. Kapur, W. Weimer, and S. Forrest. Using dynamic analysis to discover polynomial and array invariants. In *International Conference on Software Engineering (ICSE)*, pages 683–693. IEEE, 2012.

[91] T. Nguyen, D. Kapur, W. Weimer, and S. Forrest. Dig: a dynamic invariant generator for polynomial and array invariants. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 23(4):1–30, 2014.

[92] T. Nguyen, D. Kapur, W. Weimer, and S. Forrest. Using dynamic analysis to generate disjunctive invariants. In *Proceedings of the 36th International Conference on Software Engineering*, pages 608–619, 2014.

[93] T. Nguyen and K. Nguyen. Using symbolic execution to analyze linux kbuild makefiles. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 712–716. IEEE, 2020.

[94] T. Nguyen and K. Nguyen. Using symbolic execution to analyze linux kbuild makefiles. In *International Conference on Software Maintenance and Evolution*, pages 712–716. IEEE, 2020.

[95] T. Nguyen, K. Nguyen, and H. Duong. Syminfer: Inferring numerical invariants using symbolic states. In *2022 IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 197–201. IEEE, 2022.

[96] T. Nguyen, K. H. Nguyen, and M. Dwyer. Using symbolic states to infer numerical invariants. *IEEE Transactions on Software Engineering*, 2021.

[97] T. Nguyen, W. Weimer, D. Kapur, and S. Forrest. Connecting program synthesis and reachability: Automatic program repair using test-input generation. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 301–318. Springer, 2017.

[98] T.-D. Nguyen, T. Le-Cong, T. H. Nguyen, X.-B. D. Le, and Q.-T. Huynh. Toward the analysis of graph neural networks. In *2022 IEEE/ACM 44rd International Conference on Software Engineering-New Ideas and Emerging Results (ICSE-NIER)*, 2022.

[99] T. V. Nguyen, D. Ishimwe, A. Malyshev, T. Antonopoulos, and Q. S. Phan. Using dynamically inferred invariants to analyze program runtime complexity. *SEAD 2020 - Proceedings of the 3rd ACM SIGSOFT International Workshop on Software Security from Design to Deployment, Co-located with ESEC/FSE 2020*, pages 11–14, 2020.

[100] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL (T). *Journal of the ACM (JACM)*, 53(6):937–977, 2006.

[101] OVAL-group. OVAL - Branch-and-Bound-based Neural Network Verification, 2022. `https://github.com/oval-group/oval-bab`.

[102] T. Ozturk, M. Talo, E. A. Yildirim, U. B. Baloglu, O. Yildirim, and U. R. Acharya. Automated detection of covid-19 cases using deep neural networks with x-ray images. *Computers in biology and medicine*, 121:103792, 2020.

[103] J. Park, D. Winterer, C. Zhang, and Z. Su. Generative type-aware mutation for testing smt solvers. *Proceedings of the ACM on Programming Languages*, 5(OOPSLA):1–19, 2021.

[104] Project Yin-Yang. Project Yin-Yang for SMT Solver Testing, 2022. `https://testsmt.github.io`.

[105] E. Ranjan, S. Sanyal, and P. P. Talukdar. ASAP: Adaptive Structure Aware Pooling for Learning Hierarchical Graph Representations. 11 2019.

[106] Q. Rao and J. Frtunikj. Deep learning for self-driving cars: Chances and challenges. In *Proceedings of the 1st International Workshop on Software Engineering for AI in Autonomous Systems*, pages 35–38, 2018.

[107] S. Reed and N. De Freitas. Neural programmer-interpreters. *ICLR 2016*, 2016.

[108] K. Ren, T. Zheng, Z. Qin, and X. Liu. Adversarial Attacks and Defenses in Deep Learning. *Engineering*, 6(3):346–360, mar 2020.

[109] W. Ruan, X. Huang, and M. Kwiatkowska. Reachability analysis of deep neural networks with provable guarantees. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 2651–2659, 2018.

[110] A. Ruoss, M. Baader, M. Balunović, and M. Vechev. Efficient certification of spatial robustness. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 2504–2513, 2021.

[111] V. Ryvchin and O. Strichman. Faster extraction of high-level minimal unsatisfiable cores. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 174–187. Springer, 2011.

[112] P. Saha, D. Mukherjee, P. K. Singh, A. Ahmadian, M. Ferrara, and R. Sarkar. Graphcovidnet: A graph neural network based model for detecting covid-19 from ct scans and x-rays of chest. *Scientific Reports*, 11(1):1–16, 2021.

[113] R. Salay, R. Queiroz, and K. Czarnecki. An analysis of iso 26262: Using machine learning safely in automotive software. 2020.

[114] H. Salman, G. Yang, H. Zhang, C.-J. Hsieh, and P. Zhang. A convex relaxation barrier to tight robustness verification of neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[115] W. Samek, G. Montavon, A. Vedaldi, L. K. Hansen, and K.-R. Müller. *Explainable AI: interpreting, explaining and visualizing deep learning*, volume 11700. Springer Nature, 2019.

[116] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[117] D. Selsam and N. Bjørner. Guiding high-performance sat solvers with unsat-core predictions. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 336–353. Springer, 2019.

[118] D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, and D. L. Dill. Learning a SAT solver from single-bit supervision. In *International Conference on Learning Representations*, 2019.

[119] D. Shriver, S. Elbaum, and M. B. Dwyer. Dnnv: A framework for deep neural network verification. In *International Conference on Computer Aided Verification*, pages 137–150. Springer, 2021.

[120] D. Shriver, S. Elbaum, and M. B. Dwyer. Reducing dnn properties to enable falsification with adversarial attacks. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 275–287. IEEE, 2021.

[121] P. Siano, C. Cecati, H. Yu, and J. Kolbusz. Real time operation of smart grids via fcn networks and optimal power flow. *IEEE Transactions on Industrial Informatics*, 8(4):944–952, 2012.

[122] G. Singh, R. Ganvir, M. Püschel, and M. Vechev. Beyond the single neuron convex barrier for neural network certification. *Advances in Neural Information Processing Systems*, 32, 2019.

[123] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev. Fast and effective robustness certification. *Advances in neural information processing systems*, 31, 2018.

[124] G. Singh, T. Gehr, M. Püschel, and M. Vechev. Boosting robustness certification of neural networks. In *International Conference on Learning Representations*, 2018.

[125] G. Singh, T. Gehr, M. Püschel, and M. Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–30, 2019.

[126] C. Sun, V. Le, and Z. Su. Finding compiler bugs via live code mutation. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 849–863, 2016.

[127] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014*, 2014.

[128] ThanhVu Nguyen and Leonardo de Moura. Z3 Python Interface, 2022. `https://github.com/Z3Prover/z3/blob/master/src/api/python/z3/z3util.py`, last accessed July 8, 2024.

[129] ThanhVu Nguyen and Timos Antonopoulos. NLA-DIGBENCH, 2022. `https://gitlab.com/sosy-lab/benchmarking/sv-benchmarks/-/tree/main/c/nla-digbench`, last accessed July 8, 2024.

[130] A. Thota, P. Tilak, S. Ahluwalia, and N. Lohia. Fake news detection: a deep learning approach. *SMU Data Science Review*, 1(3):10, 2018.

[131] H.-D. Tran, S. Bak, W. Xiang, and T. T. Johnson. Verification of deep convolutional neural networks using imagestars. In *International conference on computer aided verification*, pages 18–42. Springer, 2020.

[132] H.-D. Tran, D. Manzanas Lopez, P. Musau, X. Yang, L. V. Nguyen, W. Xiang, and T. T. Johnson. Star-based reachability analysis of deep neural networks. In *International symposium on formal methods*, pages 670–686. Springer, 2019.

[133] H.-D. Tran, N. Pal, P. Musau, D. M. Lopez, N. Hamilton, X. Yang, S. Bak, and T. T. Johnson. Robustness verification of semantic segmentation neural networks using relaxed reachability. In *International Conference on Computer Aided Verification*, pages 263–286. Springer, 2021.

[134] H.-D. Tran, X. Yang, D. Manzanas Lopez, P. Musau, L. V. Nguyen, W. Xiang, S. Bak, and T. T. Johnson. Nnv: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *International Conference on Computer Aided Verification*, pages 3–17. Springer, 2020.

[135] C. Urban and A. Miné. A review of formal methods applied to machine learning. *arXiv preprint arXiv:2104.02466*, 2021.

[136] S. Vandenhende, S. Georgoulis, W. Van Gansbeke, M. Proesmans, D. Dai, and L. Van Gool. Multi-Task Learning for Dense Prediction Tasks: A Survey. pages 1–20, 2020.

[137] P. Veličković, R. Ying, M. Padovano, R. Hadsell, and C. Blundell. Neural execution of graph algorithms. In *International Conference on Learning Representations (ICLR)*, 2020.

[138] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. Efficient formal safety analysis of neural networks. *Advances in Neural Information Processing Systems*, 31, 2018.

[139] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. Formal security analysis of neural networks using symbolic intervals. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1599–1614, 2018.

[140] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C.-J. Hsieh, and J. Z. Kolter. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. *Advances in Neural Information Processing Systems*, 34:29909–29921, 2021.

[141] W. Weimer, T. Nguyen, C. Le Goues, and S. Forrest. Automatically finding patches using genetic programming. In *2009 IEEE 31st International Conference on Software Engineering*, pages 364–374. IEEE, 2009.

[142] N. Wetzler, M. J. Heule, and W. A. Hunt. Drat-trim: Efficient checking and trimming using expressive clausal proofs. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 422–429. Springer, 2014.

[143] D. Winterer, C. Zhang, and Z. Su. On the unusual effectiveness of type-aware operator mutations for testing smt solvers. *Proceedings of the ACM on Programming Languages*, 4(OOPSLA):1–25, 2020.

[144] D. Winterer, C. Zhang, and Z. Su. Validating smt solvers via semantic fusion. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 718–730, 2020.

[145] H. Wu, A. Ozdemir, A. Zeljic, K. Julian, A. Irfan, D. Gopinath, S. Fouladi, G. Katz, C. Pasareanu, and C. Barrett. Parallelization techniques for verifying neural networks. volume 1, pages 128–137. TU Wien Academic Press, 2020.

[146] D. Xu, D. Shriver, M. B. Dwyer, and S. Elbaum. Systematic generation of diverse benchmarks for dnn verification. In *International Conference on Computer Aided Verification*, pages 97–121. Springer, 2020.

[147] K. Xu, J. Li, M. Zhang, S. S. Du, K.-i. Kawarabayashi, and S. Jegelka. What can neural networks reason about? *ICLR 2020*, 2020.

[148] K. Xu, Z. Shi, H. Zhang, Y. Wang, K.-W. Chang, M. Huang, B. Kailkhura, X. Lin, and C.-J. Hsieh. Automatic perturbation analysis for scalable certified robustness and beyond. *Advances in Neural Information Processing Systems*, 33:1129–1141, 2020.

[149] Z. Yang, J. Shi, J. He, and D. Lo. Natural attack for pre-trained models of code. *Technical Track of ICSE 2022*, 2022.

[150] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983, 2018.

[151] Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems*, 32, 2019.

[152] A. Zeller, R. Gopinath, M. Böhme, G. Fraser, and C. Holler. The fuzzing book, 2019. `https: https://www.fuzzingbook.org`, last accessed July 8, 2024.

[153] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel. Efficient neural network robustness certification with general activation functions. *Advances in neural information processing systems*, 31, 2018.

[154] L. Zhang and S. Malik. Extracting small unsatisfiable cores from unsatisfiable boolean formula. *SAT*, 3, 2003.

[155] L. Zhang and S. Malik. Validating sat solvers using an independent resolution-based checker: Practical implementations and other applications. In *2003 Design, Automation and Test in Europe Conference and Exhibition*, pages 880–885. IEEE, 2003.

[156] T. Zhang, C. Gao, L. Ma, M. Lyu, and M. Kim. An empirical study of common challenges in developing deep learning applications. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, pages 104–115. IEEE, 2019.

[157] X. Zhang and M. Zitnik. GNNGuard: Defending Graph Neural Networks against Adversarial Attacks. *Advances in Neural Information Processing Systems*, 2020-Decem(NeurIPS), jun 2020.

[158] G. Zheng, Q. L. Le, T. Nguyen, and Q.-S. Phan. Automatic data structure repair using separation logic. *ACM SIGSOFT Software Engineering Notes*, 2018.

[159] G. Zheng, T. Nguyen, S. G. Brida, G. Regis, M. F. Frias, N. Aguirre, and H. Bagheri. Flack: Counterexample-guided fault localization for alloy models. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 637–648. IEEE, 2021.

[160] G. Zheng, T. Nguyen, S. Gutiérrez Brida, G. Regis, M. Frias, N. Aguirre, and H. Bagheri. ATR: Template-based Repair for Alloy Specifications. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 666–677, 2022.

[161] Y. Zhou, F. Wang, J. Tang, R. Nussinov, and F. Cheng. Artificial intelligence in covid-19 drug repurposing. *The Lancet Digital Health*, 2020.

[162] S. Zhu, A. Bukharin, L. Xie, M. Santillana, S. Yang, and Y. Xie. High-resolution spatio-temporal model for county-level covid-19 activity in the us. *ACM Transactions on Management Information Systems (TMIS)*, 12(4):1–20, 2021.

[163] D. Zügner, A. Akbarnejad, and S. Günnemann. Adversarial Attacks on Neural Networks for Graph Data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, volume 2019-Augus, pages 2847–2856, New York, NY, USA, jul 2018. ACM.

[164] D. Zügner, A. Akbarnejad, and S. Günnemann. Adversarial Attacks on Neural Networks for Graph Data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, volume 2019-Augus, pages 2847–2856, New York, NY, USA, jul 2018. ACM.

[165] D. Zügner and S. Günnemann. Adversarial attacks on graph neural networks via meta learning. *7th International Conference on Learning Representations, ICLR 2019*, pages 1–15, 2019.