NeuralSAT: A High-Performance Verification Tool for Deep Neural Networks

Hai Duong[1][0000−0002−3341−9794], ThanhVu Nguyen[1][0000−0002−4255−4592], and Matthew B. Dwyer[2][0000−0002−1937−1544]

[1] George Mason University, USA
{hduong22,tvn}@gmu.edu
[2] University of Virginia, USA
matthewbdwyer@virginia.edu

Abstract. Deep Neural Networks (DNNs) are increasingly deployed in critical applications, where ensuring their safety and robustness is paramount. We present $\texttt{NeuralSAT}_{\text{CAV25}}$, a high-performance DNN verification tool that uses the DPLL(T) framework and supports a wide-range of network architectures and activation functions. Since its debut in VNN-COMP'23, in which it achieved the New Participant Award and ranked 4th overall, $\texttt{NeuralSAT}_{\text{CAV25}}$ has advanced significantly, achieving second place in VNN-COMP'24. This paper presents and evaluates the latest development of $\texttt{NeuralSAT}_{\text{CAV25}}$, focusing on the versatility, ease of use, and competitive performance of the tool. $\texttt{NeuralSAT}_{\text{CAV25}}$ is available at: https://github.com/dynaroars/neuralsat.

Keywords: DNN Verification · Satisfiability Solving · VNN-COMP

1 Introduction

Deep Neural Networks (DNNs) have emerged as an effective approach for tackling challenging real-world problems. 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 sensor imperfections can result in misclassification [24,38,35,37]. These issues naturally raise the question of how DNNs should be tested, validated, and ultimately *verified* to meet the requirements of relevant robustness or safety standards [17].

To address this question, researchers have developed a wide variety of algorithmic techniques and supporting tools to verify DNNs (§5). As a result, DNN verification has become a vibrant research area, and the community has created the annual DNN verification competition (VNN-COMP) to compare different approaches, showcase the latest advances, and help shape future directions of the field [5]. The first VNN-COMP was established in 2020, and the latest iteration of the competition, VNN-COMP'24, was held with CAV in 2024.

Unlike research papers, which often focus on theoretical contributions and has a smaller evaluation scale, VNN-COMP evaluates tools based on their practical

performance on a wide range of benchmarks and properties and thus attracts the state-of-the-art in the field including: $\alpha\beta$-CROWN [30], Marabou [31] (successor of Reluplex [18]), nnenum [3], and MN-BaB [14] (successor of ERAN [13]). Among these tools, $\alpha\beta$-CROWN has been the most successful, winning the competitions four consecutive times from VNN-COMP'21 to VNN-COMP'24.

In 2023, we introduced the $\texttt{NeuralSAT}_{\text{VNNCOMP23}}$[3] verification tool in VNN-COMP'23 [6], where it ranked 4th overall and received the New Participation award (and also won the "TLL Verify Bench" benchmark category). We introduced several major improvements such as parallel DPLL(T) and neuron stabilization optimization [12] to define $\texttt{NeuralSAT}_{\text{FSE24}}$, that demonstrated competitive performance with $\alpha\beta$-CROWN on fully-connected networks. We extended $\texttt{NeuralSAT}_{\text{FSE24}}$ to support much larger set of network layer and activation function types with $\texttt{NeuralSAT}_{\text{VNNCOMP24}}$, which participated in VNN-COMP'24, where it ranked 2nd overall behind $\alpha\beta$-CROWN.

In this paper, we describe the latest version $\texttt{NeuralSAT}_{\text{CAV25}}$, which includes further extensions to optimize verification for more complex DNNs; the paper also reports on the extensions in $\texttt{NeuralSAT}_{\text{VNNCOMP24}}$ that have not been previously reported. We focus on features and engineering optimizations in $\texttt{NeuralSAT}_{\text{CAV25}}$ that are essential for creating a high-performance tool. We evaluate $\texttt{NeuralSAT}_{\text{CAV25}}$ in comparison to both $\texttt{NeuralSAT}_{\text{VNNCOMP24}}$ and the latest versions $\alpha\beta$-CROWN, and we illustrate how $\texttt{NeuralSAT}_{\text{CAV25}}$ facilitates ease of use by avoiding the complexities parameter tuning necessary in other verifiers.

Users of $\textit{NeuralSAT}_{CAV25}$. We designed $\texttt{NeuralSAT}_{\text{CAV25}}$ for (i) researchers who want to experiment with DNN verification techniques, and (ii) practitioners who want to verify their networks. For the first type of users, the DPLL(T) framework, which is carefully designed to be modular and extensible, serves as a foundation for incorporating additional algorithmic techniques from the broader SMT and DNN reasoning literature. For the second type of users, $\texttt{NeuralSAT}_{\text{CAV25}}$ works out of the box and supports various types of network architecture with minimal configuration and tuning. Our goal is to create a high-performance yet easy-to-use DNN verification tool that enables practitioners to employ state-of-the-art DNN reasoning techniques.

2 Background and Overview

2.1 The DNN verification problem

Deep Neural Network (DNN) A *deep neural network* consists of an input layer, multiple hidden layers, and an output layer. Each layer contains neurons connected to neurons in previous layers via predefined weights obtained through

[3] We use subscripts to distinguish previous versions of $\texttt{NeuralSAT}_{\text{CAV25}}$ from the version discussed in this paper. We use NeuralSAT without a subscript when we refer to the general NeuralSAT line of work.

training with data. A fully-connected (FC) layer is a layer where each neuron is connected to every neuron in the previous layer.

The output of a DNN is computed by iteratively calculating the values of neurons in each layer. Neurons in the input layer receive the input data. Neurons in the hidden layers compute their values through an *affine transformation* followed by an *activation function*, like the popular Rectified Linear Unit ($ReLU$) activation. For ReLU activation, the value of a hidden neuron y is given by $ReLU(w_1v_1 + \ldots + w_nv_n + b)$, where b is the bias parameter for y, $w_i, \ldots, w_n$ are the weights of y, $v_1, \ldots, v_n$ are the neuron values from the preceding layer, $w_1v_1 + \cdots + w_nv_n + b$ represents the affine transformation, and $ReLU(x) = \max(x, 0)$ defines the ReLU activation. A ReLU-activated neuron is *active*, if its input value is greater than zero, or *inactive*, otherwise.

DNN Verification 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 a property over the inputs of N and ϕ_{out} is a property over the outputs of N.

Modern techniques often treat the DNN verification as a *satisfiability* problem [11,31,12,30,14]. More specifically, given a formula α representing the ReLU-based DNN N and the formulae $\phi_{in} \Rightarrow \phi_{out}$ representing the property to be proved, a DNN verifier checks the satisfiability of the formula

$$\alpha \wedge \phi_{in} \wedge \overline{\phi_{out}}. \tag{1}$$

The verifier returns `unsat` if Eq. 1 is unsatisfiable, indicating that ϕ is a valid property of N, and `sat` otherwise, indicating the ϕ is not a valid property of N.

2.2 Overview of NeuralSAT

Fig. 1 gives an overview of the NeuralSAT line of work, which is modeled after the DPLL(T) framework in SMT solving [7,19]. NeuralSAT consists of standard DPLL components (non-shaded) and a theory or T-solver (shaded) dedicated for DNN reasoning.

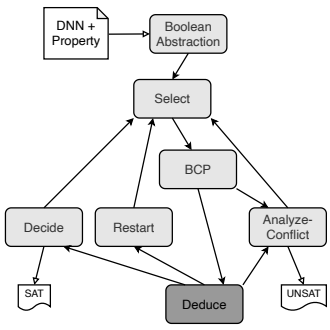

Fig. 1. NeuralSAT.

DPLL search NeuralSAT treats DNN verification as a search for an activation pattern, represented as an assignment σ which maps truth values to the variables representing the activation status of neurons (`BooleanAbstraction`). In the beginning σ is empty, and NeuralSAT uses decision heuristics to select unassigned variables (`Select`) and assigns truth values[4] to them (`Decide`). Briefly, decision heuristics in NeuralSAT work by selecting "important"

[4] As described later, NeuralSAT uses a parallel DPLL and thus will explore both branches of the decision.

neurons (or inputs) to split. `NeuralSAT` has various decision heuristics to select and will use them interchangeably depending on its optimizations. Currently, it has greedy, FSB (Filter Smart Branching) [8], and several heuristics based on neuron interval values. Particularly, NeuralSAT will change decision heuristics when a reset (discussed later) happens to create different selection orders to avoid the same orders as previous runs.

After each assignment, `NeuralSAT` infers additional assignments caused by the current assignment through Boolean constraint propagation(`BCP`). Next, it invokes the T-solver (`Deduce`) to check the feasibility of the current assignment in σ. If it is feasible, `NeuralSAT` continues to search for new assignments. Otherwise, `NeuralSAT` detects a conflict, and it learns clauses to remember and backtracks to a previous assignment (`Analyze-Conflict`).

This process repeats until `NeuralSAT` can no longer backtrack, at which point it returns `unsat`, indicating the DNN has the property. Otherwise, it finds a complete assignment for all Boolean variables (i.e., a satisfying activation pattern), and returns `sat`. The user can query for a counterexample input in the case of `sat`.

If the `NeuralSAT` search falls into a local optima, it will *restart* the search by clearing all assignments that have been made. `NeuralSAT` retains learned conflict clauses learned, to avoid reaching the same state in the subsequent search. `NeuralSAT` decides to restart when it reaches (i) a given number of verified branches, (ii) a given number of unverified branches, or (iii) exceeds a timeout. These indicate that NeuralSAT is likely stuck and needs restart.

`NeuralSAT`-specific Components `NeuralSAT` follows the standard DPLL algorithm, but includes several components specific for DNN reasoning [12].

T-Solver To check that current assignments in σ is feasible with the the formula in Eq. 1, the T-solver uses LP solving and polytope abstraction [16,34] to compute neuron bounds from the given precondition and σ, and checks the bounds are feasible with respect to the specified post-condition. Using LP solving and abstraction is standard in modern DNN verification tools [12,30,3,14]. In practice, `NeuralSAT` employs several polytopes abstraction domains to support a wide range of network types and sizes.

In addition to standard LP solving and abstraction, the T-solver implements *neuron stabilization* [12] by creating and solving custom MILP constraints to determine if a neuron is stable (i.e., it is always active or inactive). If a neuron is stable, the T-solver does not need to guess its activation status, and thus reduces the search space.

Parallel DPLL `NeuralSAT` leverages multiprocessing to parallelize its DPLL search. When assigning values to variables, `NeuralSAT` considers both options (active or inactive) for each variable, and then splits the search space into two disjoint subspaces and processes them in parallel. When a conflict is detected in one subspace, `NeuralSAT` prunes that subspace and continues the search in the

remaining subspaces. This parallelism not only speeds up the process but also facilitates information exchange such as learned clauses among search subspaces.

3 Implemented Features and Optimizations

Tab. 1. NeuralSAT's features.

Feature	Supported
Network Type	Acyclic computation graphs, e.g., Feed-forward, Residual
Layer Type	FC, CNN, MaxPool, BatchNorm, Softmax
Activation Function	ReLU, Sigmoid, Tanh, Sign, Exp
Abstract Domain	Polytope, Interval
Search Algorithm	Parallel DPLL(T)
Hardware	Multi-core CPU, GPU
Optimization	Adv. Attacks, Input splitting, Large Output Opt., MILP solving
Property	Robustness, Safety
Input	Pytorch, ONNX, VNN-LIB
Output	(sat, unsat, timeout), counter-examples

From our experience evaluating tools and participating in competitions, we found that the novelty described in research papers often does not translate to competitive performance or practical usability. Instead, the implementation details, such as being versatile, easy to use, and employing "engineering" optimizations to improve performance matter perhaps just as much. Tab. 1 shows features of NeuralSAT, many of which are often overlooked in research papers (e.g., absent in [12]) but are critical for building a long-term and high-performance tool.

Versatility The work in [12] focused on ReLU-based and fully-connected networks. NeuralSAT has since been extended to support a wide range of network architectures and activation functions. Currently, NeuralSAT works with fully connected (FC), convolutional (CNN), residual (ResNet), batch normalization (BatchNorm) networks, etc. We also support mixtures of different types, e.g., VAEs which are large residual CNN-based networks.

In addition to ReLU, NeuralSAT supports other major activation functions including sigmoid, tanh, and power. Briefly, for these non-ReLU activation functions, we split a neuron at the center of its interval. Unlike ReLU where it becomes linear after splitting, non-ReLU does not, so NeuralSAT splits a single neuron multiple times, if needed, until the problem is verified or timed out.

Note that these are also supported by other DNN verification tools such as $\alpha\beta$-CROWN though the LiRPA library [34]. However, it is straight-forward to extend NeuralSAT to support new layer or activation functions, by modifying the abstractions used in the T-solver to compute the approximation bounds of activation functions over different network layers.

Standard Input and Output Formats `NeuralSAT` supports for inputs networks in the standard ONNX format [2] and properties in VNNLIB format [26]. The output of `NeuralSAT` is reported as `unsat` (property proved), `sat` (property disproved), or `unknown` and `timeout` (property cannot be proved). `NeuralSAT` also generates counterexamples for `sat` results in text format supported by VNN-COMPs.

Fully Automatic, but Configurable An important decision in designing `NeuralSAT` is to make it fully automatic and so that for end-users it *"just works"*, perhaps even at the cost of some runtime. Users can simply apply `NeuralSAT` to check their networks and desired properties without any parameter configuration. For example, `NeuralSAT` runs on all VNN-COMP benchmarks with *zero* tuning. In contrast, top tools, such as $\alpha\beta$-`CROWN`, require significant tuning to perform effectively (more details in §4).

However, `NeuralSAT` has many settings that can be configured by the users, such as the number of threads, number of restarts, timeout, etc. These options are useful for experts who want to explore different settings and optimize the performance of `NeuralSAT` for their specific problems.

Engineering Optimizations Despite the focus on theoretical contributions in research, engineering matters! `NeuralSAT` employs various engineering optimizations to improve performance. First, like most high performing DNN verifiers, `NeuralSAT` uses adversarial attack algorithms, e.g., derivative-free sampling-based [36] and gradient-based [21] methods, to quickly find counterexamples indicating property violation. Second, `NeuralSAT` preprocesses and applies heuristics that automatically select appropriate abstractions and algorithms based on input network structures and properties. For example, `NeuralSAT` focuses on splitting the input ranges for networks with low input dimension and splitting neurons for networks with many inputs (which are the majority of real-world and VNN-COMP DNNs).

What's New? In this latest version, $\texttt{NeuralSAT}_{\text{CAV25}}$ has two new optimizations. First, for networks with large outputs (e.g., networks in "Cifar100" benchmark with 100 outputs that often cause timeout due to heavy memory usage), $\texttt{NeuralSAT}_{\text{CAV25}}$ processes multiple output constraints at once and adjusts abstraction to compute approximations that are less precise, but consume significantly less memory. Second, for networks with small ReLU-based FC layers, $\texttt{NeuralSAT}_{\text{CAV25}}$ attempts to solve the problem using MILP solving directly before using the more expensive DPLL(T) search. §4 shows the improvements of these optimizations.

Commodity Hardware `NeuralSAT` heavily leverages the power of modern hardware, including multi-core CPUs and GPUs. The parallel search in `NeuralSAT` uses multi-threading, allowing multiple search subspaces to be processed in parallel. A large part of the theory solver in `NeuralSAT` is implemented to run on GPUs, which significantly speeds up the computation of neuron bounds. While leveraging hardware is common in DNN verification, the implementation

is highly specific to the tool and requires careful engineering to achieve high performance. In VNN-COMP'24[5], NeuralSAT was one of the fastest tools, often outperforming other top competitors.

Well-Tested NeuralSAT has been rigorously tested on a wide-range of benchmarks, including those in VNN-COMPs and many more. In fact, the benchmarks in VNN-COMP are often easy for NeuralSAT, and we actively seek out more challenging benchmarks to test the tool's capabilities, through our own benchmark generation research [33,32] and collaborations with other researchers and industry partners.

Active Development NeuralSAT is actively maintained with frequent updates. If the tool does not support a specific problem or benchmark, users are encouraged to open an issue on the project's GitHub page[6], and the team will strive to provide assistance (though in practice people often send emails instead of open Github issues). While the development version of NeuralSAT is quite usable, we aim to release stable versions approximately every 6 months.

Extensibility As mentioned, NeuralSAT has many optimizations, and their addition was facilitated by the use of DPLL(T). The DPLL(T) framework in NeuralSAT is modular and extensible, consisting of a small core search algorithm and allows users to: add new decision or restart heuristics for DPLL, add new adversarial attacks in preprocessing, or extend the T-solver with additional abstraction or optimizations for DNN analysis. For example, the neuron stabilization optimization described in §2.2 is an independent function with fewer than 100 SLOCs and integrated via a hook method call into the core DPLL search. Similarly, heuristics are implemented as independent functions and can be easily replaced or extended (e.g., in current implementation decisions and restarts are less than 50 SLOC). NeuralSAT also uses the Gurobi LP solver as a black box and thus can switch to different solvers, e.g, Xpress [15], dReal [10].

Why build the solver from scratch? We did not use existing SAT or SMT solving and built NeuralSAT as a SAT solver specifically for DNN reasoning from scratch. In our experience, existing SAT/SMT solvers do not do well with DNN verification and do not scale to anything beyond the tiniest networks (in fact, none of the modern SOTA DNN tools use them). By designing our own solver, we can explore and experiment with different heuristics and optimizations without having to rely on existing ones. It also makes it much easier to add new features and optimizations as mentioned, as we do not have to worry about the limitations of existing solvers.

[5] VNN-COMP'24 no longer measures verification runtime and instead uses timeout.
[6] https://github.com/dynaroars/neuralsat

#	Tool	Score
1	$\alpha\beta$-CROWN	1200.0
2	NeuralSAT	1113.1
3	PyRAT	1000.8
4	Marabou	751.0
5	nnenum	572.5
6	NNV	530.0
7	CORA	439.5
8	NeVer2	262.3

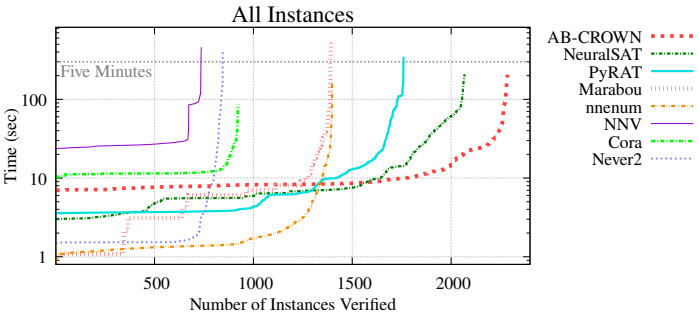

Fig. 2. VNN-COMP'24 results (of $\texttt{NeuralSAT}_{\text{VNNCOMP24}}$) [5].

4 Evaluation

4.1 VNN-COMP'24 Results

Fig. 2 summarizes the results of VNN-COMP'24 [5]. The table in the Fig. corresponds to Tab. 35 in Apdx. B of [5] and presents the overall scores and rankings of the tools. The cactus plot corresponds Fig. 28 in Apdx. B of [5] and shows tool performance on all benchmark instances. In summary, $\texttt{NeuralSAT}_{\text{VNNCOMP24}}$ ranks 2nd overall, behind $\alpha\beta$-CROWN and ahead of PyRAT.

4.2 New Results

We present the results of the latest version of $\texttt{NeuralSAT}_{\text{CAV25}}$. We also compare it with $\texttt{NeuralSAT}_{\text{VNNCOMP24}}$ and the latest version of $\alpha\beta$-CROWN[7]. As mentioned in §3, the main updates are better handling of networks with large outputs and using MILP solving. We also compare $\texttt{NeuralSAT}_{\text{CAV25}}$ with $\alpha\beta$-CROWN's default configuration, $\alpha\beta\text{-CROWN}_{\text{default}}$, to show that $\texttt{NeuralSAT}_{\text{CAV25}}$ is competitive without any parameter tuning.

Setup We reuse benchmarks and scripts for tool installation, execution, and scoring from VNN-COMP'24 [5]. In total there are 340 networks (ranging from 0.2K to 68M parameters) and 2058 properties. For, $\alpha\beta\text{-CROWN}_{\text{default}}$, we run the script provided for $\alpha\beta$-CROWN without a specific configuration (YAML) file and therefore uses its default settings. Details on the benchmarks and scoring system are available in [5] and Github repo[8].

Our experiments were run on a Linux machine with an AMD Threadripper 64-core 4.2 GHz CPU, 128 GB RAM, and an NVIDIA GeForce RTX 4090 GPU with 24 GB VRAM. Because VNN-COMP'24 used Amazon AWS instances which are different than our machine, we experimented with timeouts and settled

[7] https://github.com/Verified-Intelligence/alpha-beta-CROWN_vnncomp2024/commit/201f7401b3d8dbaddeda179939a8dc1615f8214a

[8] https://github.com/ChristopherBrix/vnncomp2024_benchmarks

on 500 seconds per instance which allowed the verifiers to achieve similar scoring performance as in VNN-COMP'24. All considered tools leverage multiprocessing and GPU processing.

Results Tab. 2 shows the results. We report the Rank (#) and % is the percentage of solved problems over all problem instances of the corresponding benchmark. The last two columns break down the number of problems each verifier was able to verify and falsify. For example, for ACAS Xu, all tools other than $\alpha\beta$-CROWN$_{\text{default}}$ were able to verify all 186 problems (139 + 47), and $\alpha\beta$-CROWN$_{\text{default}}$ was only able to solve 113 problems (78 + 35), which is 60.8% of the total problems.

Overall, $\alpha\beta$-CROWN ranks 1st, followed closely by NeuralSAT$_{\text{CAV25}}$ in 2nd, NeuralSAT$_{\text{VNNCOMP24}}$ in 3rd, and $\alpha\beta$-CROWN$_{\text{default}}$ last. For NeuralSAT$_{\text{CAV25}}$ and $\alpha\beta$-CROWN, the results are very close, with NeuralSAT$_{\text{CAV25}}$ verifying two fewer problems than $\alpha\beta$-CROWN (1296 vs. 1298) and falsifying one fewer problem (981 vs. 982). NeuralSAT$_{\text{CAV25}}$, with two new optimizations mentioned in §3, has similar performance on most benchmarks and outperforms NeuralSAT$_{\text{VNNCOMP24}}$ on the remaining ones, with the most significant improvements in "Cifar100" and "Tiny ImageNet" (due to large output optimization) and "Safe NLP" (due to MILP solving).

The results show a significant performance disparity between $\alpha\beta$-CROWN$_{\text{default}}$ and $\alpha\beta$-CROWN, with the latter having fine-tuned 10 parameters, on average, to optimize its performance for different benchmarks[9]. In contrast, NeuralSAT$_{\text{CAV25}}$ made no parameter adjustment for any benchmarks, highlighting its ease of use and potential for better performance in unseen benchmarks.

5 Related Work

The literature on DNN verification is rich and rapidly evolving (cf. [29,20]). Here we focus on tools competing in VNN-COMP'24 [5] because they are typically the state of the art and combine multiple effective DNN verification techniques.

Several tools, including NeuralSAT, belong to the BaB approach, which refines bounds computed for subproblems and then splits, or branches, them into subproblems that are solved separately. Marabou [31] (the successor of the popular Reluplex work [18]) encodes verification as constraint problems and uses parallelized split-and-conquer techniques for efficiency. nnenum [3] uses hidden BaB with star sets and several types of zonotope abstractions and focuses strictly on ReLU networks. The mentioned $\alpha\beta$-CROWN [30] combines GPU-accelerated linear bound propagation with advanced BaB techniques, such as cutting planes and neuron splitting, to scale to large networks. While both NeuralSAT and $\alpha\beta$-CROWN use BaB and GPU acceleration, they differ widely in everything else,

[9] https://github.com/Verified-Intelligence/alpha-beta-CROWN_vnncomp2024/blob/master/complete_verifier/exp_configs/vnncomp24/ partially consists of VNN-COMP'24 runscripts of $\alpha\beta$-CROWN, which use different configurations (in yaml) on different benchmarks.

Tab. 2. Results over VNN-COMP'24 Benchmarks

Benchmark	#	Tool	%	Verify	Falsify
ACAS Xu	1	$\alpha\beta$-CROWN	100.0%	**139**	**47**
	1	NeuralSAT$_{\text{CAV25}}$	100.0%	**139**	**47**
	1	NeuralSAT$_{\text{VNNCOMP24}}$	100.0%	**139**	**47**
	4	$\alpha\beta$-CROWN$_{\text{default}}$	60.8%	78	35
Cgan	1	$\alpha\beta$-CROWN	100.0%	**8**	**13**
	1	NeuralSAT$_{\text{CAV25}}$	100.0%	**8**	**13**
	1	NeuralSAT$_{\text{VNNCOMP24}}$	100.0%	**8**	**13**
	4	$\alpha\beta$-CROWN$_{\text{default}}$	33.3%	0	7
Cifar100	1	$\alpha\beta$-CROWN	77.5%	**123**	**32**
	2	NeuralSAT$_{\text{CAV25}}$	76.5%	122	31
	3	$\alpha\beta$-CROWN$_{\text{default}}$	71.0%	110	**32**
	4	NeuralSAT$_{\text{VNNCOMP24}}$	64.5%	98	31
Collins Rul CNN	1	$\alpha\beta$-CROWN	100.0%	**30**	**32**
	1	$\alpha\beta$-CROWN$_{\text{default}}$	100.0%	**30**	**32**
	1	NeuralSAT$_{\text{CAV25}}$	100.0%	**30**	**32**
	1	NeuralSAT$_{\text{VNNCOMP24}}$	100.0%	**30**	**32**
Cora	1	$\alpha\beta$-CROWN	43.9%	**24**	**134**
	1	$\alpha\beta$-CROWN$_{\text{default}}$	43.9%	**24**	**134**
	1	NeuralSAT$_{\text{CAV25}}$	43.9%	**24**	**134**
	1	NeuralSAT$_{\text{VNNCOMP24}}$	43.9%	**24**	**134**
Dist Shift	1	$\alpha\beta$-CROWN	100.0%	**64**	**8**
	1	NeuralSAT$_{\text{CAV25}}$	100.0%	**64**	**8**
	3	NeuralSAT$_{\text{VNNCOMP24}}$	98.6%	63	**8**
	4	$\alpha\beta$-CROWN$_{\text{default}}$	94.4%	60	**8**
Linearize NN	1	$\alpha\beta$-CROWN	100.0%	**59**	**1**
	1	NeuralSAT$_{\text{CAV25}}$	100.0%	**59**	**1**
	1	NeuralSAT$_{\text{VNNCOMP24}}$	100.0%	**59**	**1**
	4	$\alpha\beta$-CROWN$_{\text{default}}$	68.3%	40	**1**
Meta Room	1	$\alpha\beta$-CROWN	98.0%	**91**	**7**
	1	NeuralSAT$_{\text{CAV25}}$	98.0%	**91**	**7**
	1	NeuralSAT$_{\text{VNNCOMP24}}$	98.0%	**91**	**7**
	4	$\alpha\beta$-CROWN$_{\text{default}}$	0.0%	0	0
Nn4sys	1	$\alpha\beta$-CROWN	100.0%	**194**	0
	1	NeuralSAT$_{\text{CAV25}}$	100.0%	**194**	0
	1	NeuralSAT$_{\text{VNNCOMP24}}$	100.0%	**194**	0
	4	$\alpha\beta$-CROWN$_{\text{default}}$	4.1%	8	0
Safe NLP	1	$\alpha\beta$-CROWN	98.1%	**411**	**648**
	1	NeuralSAT$_{\text{CAV25}}$	98.1%	**411**	**648**
	3	$\alpha\beta$-CROWN$_{\text{default}}$	96.9%	401	646
	4	NeuralSAT$_{\text{VNNCOMP24}}$	94.3%	378	640
Tiny ImageNet	1	$\alpha\beta$-CROWN	91.5%	**140**	**43**
	2	NeuralSAT$_{\text{CAV25}}$	91.0%	139	**43**
	3	$\alpha\beta$-CROWN$_{\text{default}}$	89.5%	136	**43**
	4	NeuralSAT$_{\text{VNNCOMP24}}$	72.5%	102	**43**
TLL Verify Bench	1	$\alpha\beta$-CROWN	100.0%	**15**	**17**
	1	NeuralSAT$_{\text{CAV25}}$	100.0%	**15**	**17**
	1	NeuralSAT$_{\text{VNNCOMP24}}$	100.0%	**15**	**17**
	4	$\alpha\beta$-CROWN$_{\text{default}}$	65.6%	5	16
Overall	1	$\alpha\beta$-CROWN	88.8%	**1298**	**982**
	2	NeuralSAT$_{\text{CAV25}}$	88.7%	1296	981
	3	NeuralSAT$_{\text{VNNCOMP24}}$	84.7%	1201	973
	4	$\alpha\beta$-CROWN$_{\text{default}}$	71.9%	892	954

e.g., heuristics and optimizations, with the main distinctions being NeuralSAT prioritize splitting unstable neurons and have strategies borrowed from SAT solving such as restart when it reaches local optima [12].

Other tools use reachability analysis, which overapproximates reachable states to verify properties. CORA [1] employs zonotopes for non-convex enclosures for open-loop and closed-loop verification in control systems. NeVer2 [9] focuses on ReLU-based feedforward networks by using an abstraction-refinement algorithm with symbolic bounds propagation. PyRAT [23] uses abstract interpretation with many domains including intervals, zonotopes, and polyhedra to compute sound overapproximations of reachable states to verify safe and robustness properties. NNV [28] focuses on verifying network-based control systems by integrating the star-set domain [27] with iteratively refinement for precise reachability analysis.

NeuralSAT achieves BaB through its DPLL(T) framework, which provides a strong algorithmic foundation and the flexibility to explore new heuristics and optimizations. NeuralSAT also delivers competitive performance out-of-the-box–an advantage over tools that require significant parameter tuning for good performance.

6 Conclusion and Future Work

NeuralSAT has quickly evolved into a leading performer in DNN verification, achieving similar performance to established competitors like $\alpha\beta$-CROWN. By adopting modular and extensible designs, parallel DPLL(T) search, and advanced optimizations, NeuralSAT performs competitively across a diverse set of benchmarks, demonstrating its robustness and scalability. Its out-of-the-box usability, combined with its potential for further optimization and customization, make it an attractive choice for both researchers and practitioners.

Maintaining competitiveness in the world of rapid advancements requires continuous innovation in both algorithmic research and engineering advancements. We are exploring both algorithmic research, such as compositional reasoning [22,25], which decomposes large verification to more manageable subproblems, and decision heuristics from DPLL, such as VMTF (Variable Move-to-Front) [4], which prioritize variables involved in learned conflict clauses to improve search efficiency, and engineering improvements, such as enhancing parallelization and supporting multi-GPU hardware acceleration.

Acknowledgments

This material is based in part upon work supported by the National Science Foundation under grant numbers 2019239, 2129824, 2200621, 2217071, 2238133, 2319131, 2422036, and by an Amazon Research Award.

References

1. Althoff, M.: An introduction to cora 2015. In: Proc. of the workshop on applied verification for continuous and hybrid systems. pp. 120–151 (2015). https://doi.org/10.29007/zbkv
2. Bai, J., Lu, F., Zhang, K.: ONNX Open neural network exchange, https://onnx.ai
3. Bak, S.: nnenum: Verification of ReLU Neural Networks with Optimized Abstraction Refinement. In: NASA Formal Methods Symposium. pp. 19–36. Springer (2021). https://doi.org/10.1007/978-3-030-76384-8_2
4. Biere, A., Fr"ohlich, A.: Evaluating cdcl variable scoring schemes. In: Theory and Applications of Satisfiability Testing–SAT 2015: 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings 18. pp. 405–422. Springer (2015). https://doi.org/10.1007/978-3-319-24318-4_29
5. Brix, C., Bak, S., Johnson, T.T., Wu, H.: The Fifth International Verification of Neural Networks Competition (VNN-COMP 2024): Summary and Results. arXiv preprint arXiv:2412.19985 (2024). https://doi.org/10.48550/arXiv.2412.19985
6. Brix, C., Bak, S., Liu, C., Johnson, T.T.: The Fourth International Verification of Neural Networks Competition (VNN-COMP 2023): Summary and Results (2023). https://doi.org/10.48550/arXiv.2312.16760
7. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Communications of the ACM **5**(7), 394–397 (1962), https://dl.acm.org/doi/10.1145/368273.368557
8. De Palma, A., Bunel, R., Desmaison, A., Dvijotham, K., Kohli, P., Torr, P.H., Kumar, M.P.: Improved branch and bound for neural network verification via lagrangian decomposition. arXiv preprint arXiv:2104.06718 (2021). https://doi.org/10.48550/arXiv.2104.06718
9. Demarchi, S., Guidotti, D., Pulina, L., Tacchella, A.: Never2: learning and verification of neural networks. Soft Computing **28**(19), 11647–11665 (2024). https://doi.org/10.1007/s00500-024-09907-5
10. dreal: An SMT Solver for Nonlinear Theories of Reals (2024), https://dreal.github.io/
11. Duong, H., Nguyen, T., Dwyer, M.: A DPLL(T) Framework for Verifying Deep Neural Networks. arXiv preprint arXiv:2307.10266 (2024). https://doi.org/10.48550/arXiv.2307.10266
12. Duong, H., Xu, D., Nguyen, T., Dwyer, M.B.: Harnessing neuron stability to improve dnn verification. Proceedings of the ACM on Software Engineering **1**(FSE), 859–881 (2024). https://doi.org/10.1145/3643765
13. ETH-SRI: ETH Robustness Analyzer for Deep Neural Networks (2021), https://github.com/eth-sri/eran
14. Ferrari, C., Mueller, M.N., Jovanović, N., Vechev, M.: Complete Verification via Multi-Neuron Relaxation Guided Branch-and-Bound. In: International Conference on Learning Representations (2022). https://doi.org/10.48550/arXiv.2205.00263
15. FICO: Xpress Optimization (2024), https://www.fico.com/en/products/fico-xpress-optimization
16. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2022), https://www.gurobi.com
17. Huang, X., Kroening, D., Ruan, W., Sharp, J., Sun, Y., Thamo, E., Wu, M., Yi, X.: A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. Computer Science Review **37**, 100270 (2020). https://doi.org/10.1016/j.cosrev.2020.100270

18. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. In: International Conference on Computer Aided Verification. pp. 97–117. Springer (2017). https://doi.org/10.1007/978-3-319-63387-9_5
19. Kroening, D., Strichman, O.: Decision procedures. Springer (2008), https://dl.acm.org/doi/10.5555/1391237
20. Liu, C., Arnon, T., Lazarus, C., Strong, C., Barrett, C., Kochenderfer, M.J., et al.: Algorithms for verifying deep neural networks. Foundations and Trends® in Optimization **4**(3-4), 244–404 (2021). https://doi.org/10.1561/2400000035
21. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. arXiv preprint arXiv:1706.06083 (2017), https://hdl.handle.net/1721.1/137496
22. Misra, J., Chandy, K.M.: Proofs of networks of processes. IEEE transactions on software engineering pp. 417–426 (1981). https://doi.org/10.1109/TSE.1981.230844
23. PyRAT: A tool to analyze the robustness and safety of neural networks (2024), https://pyrat-analyzer.com/
24. Ren, K., Zheng, T., Qin, Z., Liu, X.: Adversarial Attacks and Defenses in Deep Learning. Engineering **6**(3), 346–360 (2020). https://doi.org/10.1016/j.eng.2019.12.012
25. Stark, E.W.: A proof technique for rely/guarantee properties. In: Foundations of Software Technology and Theoretical Computer Science: Fifth Conference, New Delhi, India December 16–18, 1985 Proceedings 5. pp. 369–391. Springer (1985). https://doi.org/10.1007/3-540-16042-6_21
26. Tacchella, A., Pulina, L., Guidotti, D., Demarchi, S.: The international benchmarks standard for the Verification of Neural Networks (2023), https://www.vnnlib.org/
27. Tran, H.D., Manzanas Lopez, D., Musau, P., Yang, X., Nguyen, L.V., Xiang, W., Johnson, T.T.: Star-Based Reachability Analysis of Deep Neural Networks. In: International symposium on formal methods. pp. 670–686. Springer (2019). https://doi.org/10.1007/978-3-030-30942-8_39
28. Tran, H.D., Yang, X., Manzanas Lopez, D., Musau, P., Nguyen, L.V., Xiang, W., Bak, S., Johnson, T.T.: Nnv: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In: International Conference on Computer Aided Verification. pp. 3–17. Springer (2020). https://doi.org/10.1007/978-3-030-53288-8_1
29. Urban, C., Miné, A.: A review of formal methods applied to machine learning. arXiv preprint arXiv:2104.02466 (2021). https://doi.org/10.48550/arXiv.2104.02466
30. Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C.J., Kolter, J.Z.: Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Complete and Incomplete Neural Network Robustness Verification. Advances in Neural Information Processing Systems **34**, 29909–29921 (2021). https://doi.org/10.48550/arXiv.2103.06624
31. Wu, H., Isac, O., Zeljić, A., Tagomori, T., Daggitt, M., Kokke, W., Refaeli, I., Amir, G., Julian, K., Bassan, S., et al.: Marabou 2.0: a versatile formal analyzer of neural networks. In: International Conference on Computer Aided Verification. pp. 249–264. Springer (2024). https://doi.org/10.48550/arXiv.2401.14461
32. Xu, D., Mozumder, N.J., Duong, H., Dwyer, M.: Training for verification: Increasing neuron stability to scale DNN verification. In: Tools and Algorithms for the Construction and Analysis of Systems - 30th International Conference, TACAS 2024, Held as Part of the European Joint Conferences on Theory and

Practice of Software, ETAPS. p. to appear. Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_2
33. Xu, D., Shriver, D., Dwyer, M.B., Elbaum, S.: Systematic Generation of Diverse Benchmarks for DNN Verification. In: International Conference on Computer Aided Verification. pp. 97–121. Springer (2020). https://doi.org/10.1007/978-3-030-53288-8_5
34. Xu, K., Shi, Z., Zhang, H., Wang, Y., Chang, K.W., Huang, M., Kailkhura, B., Lin, X., Hsieh, C.J.: Automatic perturbation analysis for scalable certified robustness and beyond. Advances in Neural Information Processing Systems **33**, 1129–1141 (2020), https://dl.acm.org/doi/10.5555/3495724.3495820
35. Yang, Z., Shi, J., He, J., Lo, D.: Natural attack for pre-trained models of code. In: Proceedings of the 44th International Conference on Software Engineering. pp. 1482–1493 (2022). https://doi.org/10.1145/3510003.3510146
36. Yu, Y., Qian, H., Hu, Y.Q.: Derivative-free optimization via classification. In: Thirtieth AAAI Conference on Artificial Intelligence (2016), https://dl.acm.org/doi/10.5555/3016100.3016218
37. Zhang, T., Gao, C., Ma, L., Lyu, M., Kim, M.: An empirical study of common challenges in developing deep learning applications. In: 2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE). pp. 104–115. IEEE (2019). https://doi.org/10.1109/ISSRE.2019.00020
38. Zügner, D., Akbarnejad, A., Günnemann, S.: Adversarial Attacks on Neural Networks for Graph Data. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. vol. 2019-Augus, pp. 2847–2856. ACM, New York, NY, USA (jul 2018). https://doi.org/10.1145/3219819.3220078