NeuralSAT: Scaling Constraint Solving for DNN Verification (Competition Contribution)

Hai Duong[1] and ThanhVu Nguyen[1]

George Mason University, USA

Abstract. We present NeuralSAT, a DNN verification tool based on the DPLL(T) framework in SAT solving with conflict clause learning. NeuralSAT participated in VNN-COMP'23 and VNN-COMP'24, with recent improvements such as parallel DPLL(T) and neuron stabilization optimizations. The theoretical foundations and algorithmic details of NeuralSAT are described in prior work, and this paper focuses on the engineering aspects of NeuralSAT, including its design, configuration, and performance in the context of the VNN-COMP evaluation framework. NeuralSAT is available at: https://github.com/dynaroars/neuralsat.

Keywords: DNN Verification · Satisfiability Solving · VNN-COMP

1 Introduction

Deep learning systems are increasingly being deployed in safety-critical domains such as autonomous driving and healthcare. However, like traditional software systems, neural networks have "bugs" and vulnerable to attacks—raising concerns on their deployment in the real-world. Deep neural network (DNN) verification has emerged as a promising research direction to address this gap, resulting in a wide variety of algorithmic techniques and supporting tools.

To foster research progress and enable fair comparison between DNN verification tools, the International Verification of Neural Networks Competition (VNN-COMP) was established in 2020 and has been held annually as a co-located event CAV. VNN-COMP [2] provides a standardized evaluation framework, including common formats for neural networks and specifications, a uniform benchmarking infrastructure on AWS cloud instances, and automated pipelines for tool installation and evaluation. Since its inception, the competition has attracted many state-of-the-art in the field.

In 2023, we introduced NeuralSAT, a DNN verification tool based on the DPLL(Davis-Putnam-Logemann-Loveland) with theory T framework [3,8], and submitted it to VNN-COMP'23. The following year, NeuralSAT was updated with features such as parallel DPLL(T) and neuron stabilization optimization [5], and participated again in VNN-COMP'24. The NeuralSAT's algorithm, along with optimizations and an in-depth evaluation, is presented in [4,5]. This contribution paper focuses on NeuralSAT's implementation and settings in the context of VNN-COMP.

2 Overview of NeuralSAT

Fig. 1 summarizes the DPLL(T) framework [3,8] that NeuralSAT implements. It consists of standard DPLL components (non-shaded) and a theory solver (shaded) dedicated for DNN reasoning.

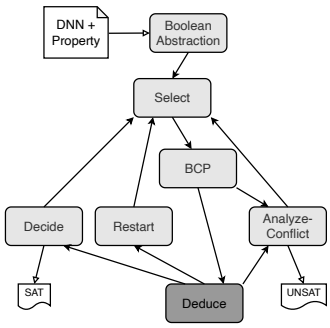

Fig. 1: NeuralSAT.

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). Initially σ is empty, and NeuralSAT uses decision heuristics to select unassigned variables (Select) and assigns truth values to them (Decide). 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, i.e., 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 search falls into a local optima, NeuralSAT will restart by clearing all assignments that have been made. NeuralSAT retains learned conflict clauses learned, to avoid reaching the same state in the subsequent search.

Note that NeuralSAT leverages multiprocessing to parallelize its DPLL search. When assigning values to variables, NeuralSAT considers both options 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.

Theory (T)-Solver To check that current assignments in σ is feasible, the T-solver uses LP solving and polytope abstraction [7,16] 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 [19,6,4,11,15,18,5]. However, the T-solver in NeuralSAT also implements *neuron stabilization* 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.

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
Input	Pytorch, ONNX, VNN-LIB
Output	(sat, unsat, timeout), counter-examples
Property	Robustness, Safety
Search Algorithm	Parallel DPLL(T)
Abstract Domain	Polytope, Interval
Hardware	Multi-core CPU, GPU
Optimization	Adv. Attacks, Input splitting, Large Output Opt., MILP solving

3 Implementation, Features, and Limitations

NeuralSAT is implemented in Python, and uses Gurobi [7] for linear programming and the LiRPA library [16] for computing neuron bounds and other abstractions. Tab. 1 highlights the key implementation features.

3.1 Features

Here we discuss several design decisions and features of NeuralSAT that we believe are important for competition and practical use.

Fully Automatic, Yet Configurable A key design decision was to make NeuralSAT fully automatic and *"just works"* for end users, even at some runtime cost. Invoking NeuralSAT is as simple as running a single command:

```
python3 main.py -net <n> -spec <p>
```

where <n> is an ONNX network file, and <p> is a VNNLIB specification file. During VNN-COMP, NeuralSAT was run on all benchmarks using a single default configuration, requiring no parameter tuning. Expert users can optionally adjust parameters such as the number of threads and timeout durations via command-line flags.

While NeuralSAT is fully automatic by default, it has a wide-range of configurable parameters, such as the number of threads, restart limits, and timeout durations. These options, which can be set via command-line flags or configuration files, allow expert users to fine-tune NeuralSAT's performance to suit their specific needs.

Engineering Optimizations. NeuralSAT integrates several practical optimizations to enhance verification speed and scalability. It employs adversarial attack strategies, such as derivative-free sampling [17] and gradient-based [9] methods,

to quickly identify counterexamples when properties are violated. The tool also includes preprocessing logic and automated heuristics to select suitable abstractions or reasoning algorithms according to the structure and dimensionality of the input network. For example, `NeuralSAT` prioritizes input range splitting for low-dimensional inputs and neuron splitting for high-dimensional cases. When working with networks with large output spaces or small ReLU-based fully connected layers, `NeuralSAT` adaptively adjusts abstraction granularity or applies MILP solving to improve efficiency.

3.2 Limitations

`NeuralSAT` has several limitations. First, it does not support other architectures with cycles, such as graph neural networks (GNNs). Second, it only supports properties that can be expressed in the VNN-LIB format, which at current time mainly consists of safety and robustness properties. Third, `NeuralSAT` depends on high-performance hardware, such as multi-core CPUs and GPUs, and thus performs poorly on low-end machines like standard laptops or desktops. Finally, it relies on Gurobi, a proprietary, general-purpose LP solver that does not leverage GPUs, which are commonly used in DNN reasoning, and therefore can become the bottleneck when verifying large networks.

4 VNN-COMP Participation

We summarize the setup and configuration of `NeuralSAT` in VNN-COMP'24. The full details including runscripts and results are available in the official VNN-COMP Github repo [13,14] and report [1].

Benchmark Participation `NeuralSAT` competes in all regular benchmark categories of VNN-COMP. For example, in VNN-COMP'24, it was evaluated on all 12 standard benchmarks [13], including ACAS Xu, cGAN, Cifar100, Collins Rul CNN, Cora, DistShift, LinearizeNN, MetaRoom, NN4Sys, SafeNLP, TinyImageNet, and TLLVerifyBench. These benchmarks cover a diverse set of neural networks, ranging from models with a single input (e.g., 1 for NN4Sys) to those with high-dimensional inputs (e.g., 9408 for TinyImageNet), and from small networks with only a few thousand parameters (e.g., 4K for SafeNLP) to large models with tens of millions of parameters (e.g., 68M for cGAN). In total, the regular track features 2,567 benchmark instances, where an instance is a pair of a network and a verification property.

Configuration As described in §3.1, `NeuralSAT` is designed to work out of the box. In VNN-COMP'24, `NeuralSAT` was run using a default configuration, with examples of major parameters set as follows:

```
-batch 1000          -max_hidden_visited_branches 20000
-attack_interval 10  -mip_tightening_topk  64
```

#	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

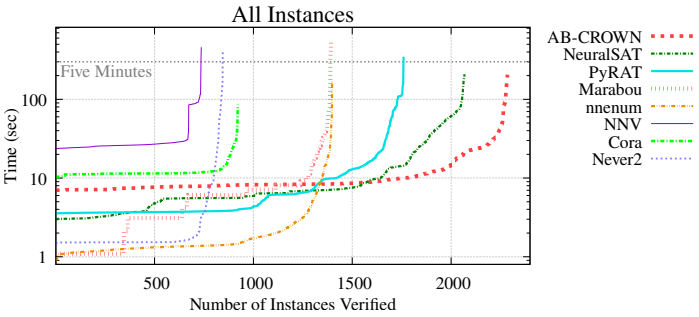

Fig. 2: VNN-COMP'24 updated results [1].

The option `-batch` specifies the number of branches to explore in parallel, `-max_hidden_visited_branches` indicates the maximum number of hidden-layer branches to explore before triggering a restart, `-attack_interval` specifies how often to apply adversarial attacks, and `-mip_tightening_topk` sets the number of neurons to stabilize during the verification process (set to the number of threads available on the CPU).

Results Fig. 2 summarizes the results[1] of VNN-COMP'24 [1]. The table, which corresponds to Tab. 35 in Apdx. B of [1], presents the overall rankings and scores of participating tools. The cactus plot, corresponds Fig. 29 in Apdx. B of [1], shows performance of tools on all benchmark instances.

5 Conclusion and Ongoing Work

The DPLL-based NeuralSAT DNN verification tool demonstrated competitive performance in recent VNN-COMPs, achieving high scores across benchmark categories. The project is open source under the MIT license and available at https://github.com/dynaroars/neuralsat. We weclome contributions from the community and encourage users to report issues or suggest improvements via GitHub. Ongoing work includes new search strategies and engineering optimizations, e.g., compositional reasoning [10,12].

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

[1] As is standard in VNN-COMP, after the initial results are released, authors are given the opportunity to review and report errors. In VNN-COMP'24, a problem with the competition's output parsing script resulted in NeuralSAT being incorrectly ranked last. After this issue, which also affected other tools, was identified and fixed within the designated review period, the official VNN-COMP results were updated, with NeuralSAT placed 2nd overall. The issues, corrections, and updated results are fully documented in the final VNN-COMP'24 report (e.g., see [1, Apdx. B])

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. 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
2. 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
3. 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
4. Duong, H., Nguyen, T., Dwyer, M.B.: Neuralsat: A high-performance verification tool for deep neural networks. In: International Conference on Computer Aided Verification. p. to appear (2025)
5. 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
6. 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
7. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2022), https://www.gurobi.com
8. Kroening, D., Strichman, O.: Decision procedures. Springer (2008), https://dl.acm.org/doi/10.5555/1391237
9. 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
10. 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
11. PyRAT: A tool to analyze the robustness and safety of neural networks (2024), https://pyrat-analyzer.com/
12. 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
13. VNN-COMP 2024: VNN-COMP 2024 Regular Benchmarks (2024), https://github.com/ChristopherBrix/vnncomp2024_benchmarks
14. VNN-COMP 2024: VNN-COMP 2024 Results (2024), https://github.com/VNN-COMP/vnncomp2024_results
15. 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
16. 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
17. 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
18. Zhang, H., Wang, S., Xu, K., Li, L., Li, B., Jana, S., Hsieh, C.J., Kolter, J.Z.: General cutting planes for bound-propagation-based neural network verification. Proceedings of the 36th International Conference on Neural Information Processing Systems (2022), https://dl.acm.org/doi/10.5555/3600270.3600391
19. Zhou, D., Brix, C., Hanasusanto, G.A., Zhang, H.: Scalable neural network verification with branch-and-bound inferred cutting planes. arXiv preprint arXiv:2501.00200 (2024)