n2v: Neural Network Verification in Python (Competition Contribution)

Samuel Sasaki, Ben Wooding, Hanchen David Wang, Anne M. Tumlin, Meiyi Ma, and Taylor T. Johnson

Vanderbilt University, Nashville, TN 37212, USA
{samuel.sasaki,ben.wooding,hanchen.wang.1,anne.m.tumlin,meiyi.ma,taylor.johnson}@vanderbilt.edu

Abstract. We present n2v, an open-source Python toolbox for neural network verification. n2v is a successor to the MATLAB-based NNV tool, reimplemented in Python with native PyTorch integration to lower the barrier to adoption for the deep learning and verification communities. Like NNV, n2v supports exact and over-approximate reachability using star sets, zonotopes, and hyperrectangles. In addition, n2v contributes probabilistic verification via conformal inference, PGD-based falsification, and a batched HiGHS-backed linear-programming pipeline. We describe n2v's architecture and the verification workflow it uses for the Verification of Neural Networks Competition (VNN-COMP).

1 Introduction

n2v is a Python toolbox for neural network (NN) verification, developed as a successor to the MATLAB-based NNV tool [13,6]. Its core is a layer-wise reachability engine over convex set representations — star sets [10], zonotopes [2], and hyperrectangles — that computes exact reachable sets for NNs with piecewise-linear activations and sound over-approximations for both piecewise-linear and smooth nonlinearities such as sigmoid and tanh.

Beyond NNV's core reachability, n2v contributes: *(i)* probabilistic verification via conformal inference [3] with formal probabilistic coverage guarantees, used selectively on benchmarks where sound reachability is intractable; *(ii)* projected gradient descent (PGD)-based falsification, complementing NNV's random-sampling counterexample search; and *(iii)* a batched HiGHS [4]-backed linear-programming (LP) pipeline with thread-parallel workers, replacing per-objective LP calls.

Why Python? Our decision to reimplement NNV in Python is driven by three concerns. *(i) Ecosystem alignment*: the deep learning community is overwhelmingly Python-based, and a PyTorch-native verifier eliminates the MATLAB $\leftrightarrow$ ONNX round-trip required to bring trained models into NNV. n2v consumes `torch.nn.Module` instances directly, so the same model object used during training and evaluation can be passed to the verifier. *(ii) Accessibility*: NNV requires

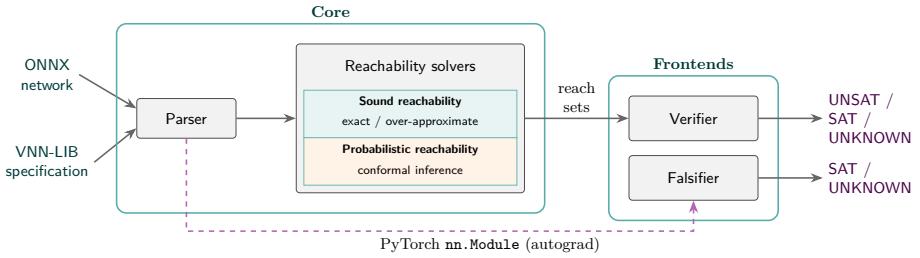

Fig. 1. Overview of the n2v architecture. ONNX networks and VNN-LIB specifications are parsed into the core where reachable sets are computed via sound (exact / over-approximate) or probabilistic methods. Verification frontends consume these reach sets and produce a per-instance verdict, and the falsifier reuses the original PyTorch model (dashed arrow) for autograd-based PGD.

a MATLAB license and depends on several MATLAB toolboxes (Computer Vision, Deep Learning, Optimization, Parallel Computing, Statistics, *etc.*). In comparison, n2v has no proprietary dependencies and installs through `pip`. *(iii) Lower onboarding cost*: the multi-step MATLAB setup is replaced with a clean install process (cloning the repository and `pip install`ing the dependencies) lowering the barrier for new users in courses, on cloud workers, or in continuous-integration pipelines.

Within the Python ecosystem, NN verification has been pursued primarily through bound propagation (e.g., α,β-CROWN [17,14], `auto_LiRPA` [16], ERAN [8]) and SMT search (Marabou [15]). n2v brings reachability-based verification to this ecosystem, centering on star sets with ImageStar [9] extensions for convolutional networks and probabilistic reachability via conformal inference for cases where sound reachability is intractable.

A side benefit of accepting PyTorch's `nn.Module` as input is that the same model object drives both autograd-based PGD falsification and reachability analysis, which is performed by walking a `torch.fx` symbolic trace of the module. There is therefore no verifier-side model translation analogous to NNV's `matlab2nnv`.

2 Competition Approach

For VNN-COMP 2026, n2v participates in all regular-track benchmarks supported by our ONNX importer (`onnx2torch`[1]) and in as many extended-track benchmarks as the importer can convert. Described below is the four-step procedure generally followed for each benchmark in the competition, with exceptions.

Step 1. *Parsing.* The ONNX network is loaded into a PyTorch `GraphModule` via `onnx2torch`, and the VNN-LIB [1,7] specification is parsed into an input

[1] Forked from https://github.com/ENOT-AutoDL/onnx2torch

Table 1. Comparison of `n2v` and NNV on the ACAS Xu benchmark. `n2v` is run with the NNV-style pipeline (falsification → approximate reachability → exact reachability).

Tool	SAT	UNSAT	Timeout	Error	Solved
`n2v`	38	85	63	0	**123**/186
NNV	39	71	75	1	110/186

set (typically a Star representing a hyperrectangle) and an output property encoded as a disjunction of half-space conjunctions.

Step 2. *Falsification.* Before invoking reachability, `n2v` searches for counterexamples by uniform random sampling followed by PGD with multiple restarts on the loaded PyTorch model, optimizing a property loss via autograd. When a counterexample is found, `n2v` returns `SAT` together with the witness input.

Step 3. *Approximate reachability.* If falsification fails to produce a counterexample, `n2v` computes a sound over-approximation of the output reachable set by applying the triangle relaxation at ReLU layers and an S-curve relaxation at sigmoid and tanh layers without case splitting [12]. When this over-approximation is disjoint from the unsafe region, the instance is reported `UNSAT`.

Step 4. *Exact reachability.* Otherwise `n2v` invokes exact reachability by case-splitting at unstable piecewise-linear units, which formally guarantees correct `UNSAT`/`SAT` decisions [10] when case-splitting terminates within the per-instance time budget. If exact reachability does not complete in time, `n2v` returns `TIMEOUT`. A zonotope pre-pass [11] optionally precomputes intermediate bounds to eliminate stable neurons before splitting.

† *Probabilistic reachability.* On benchmarks where sound reachability is intractable within the per-instance time budget, `n2v` can be configured to use probabilistic verification via conformal inference. A calibration set is drawn from the benchmark's input hyperrectangle and is used to characterize the true output reach set with some probabilistic guarantee. `n2v` returns `UNSAT` when the probabilistic reach set is disjoint from the unsafe region and `UNKNOWN` when it is not, trading formal soundness for tractability.

Table 1 compares `n2v` against NNV on the ACAS Xu benchmark used at VNN-COMP. The results of each individual instance have been checked for soundness issues of which there are none. However, the NNV results shown here are from last year's iteration of VNN-COMP [5] and therefore these experiments have not been run on equivalent hardware. `n2v` performs significantly better with 13 more instances verified than NNV.

3 Tool Architecture

`n2v` is implemented in Python on top of the PyTorch and SciPy ecosystems. Its architecture (Figure 1) is organized as three layers and realizes the procedure of Section 2. An input layer parses ONNX networks and VNN-LIB specifications

into the data structures n2v reasons over. A core reachability dispatcher walks the parsed network layer by layer over a library of convex set representations. Verification frontends consume the resulting reachable sets to produce a per-instance verdict, and they reuse the original PyTorch model for autograd-based falsification.

Inputs. n2v accepts neural networks in ONNX and specifications in VNN-LIB, the two formats used throughout VNN-COMP. ONNX networks are first traced into a PyTorch `GraphModule` by `onnx2torch`. ONNX operations that lack a direct `nn.Module` equivalent[2] are handled by an in-tree graph executor. VNN-LIB specifications are parsed in-tree into an input set together with a disjunctive half-space output property.

Core. At the core of n2v is a reachability dispatcher that operates layer by layer over a library of convex sets. All set types share a uniform interface for construction from bounds, affine maps, range queries, containment checks, and sampling. n2v relies on three set representations. *Stars*, defined by $x = c + V\alpha$ with $C\alpha \leq d$, support both exact and approximate reachability. *Zonotopes*, defined by $x = c + V\alpha$ with $\alpha \in [-1, 1]$, provide fast bound pre-computation. *Hyperrectangles* parse VNN-LIB inputs. Convolutional neural networks additionally use *ImageStars* [9], which extend Stars to four-dimensional tensors resulting in more memory-efficient reachability on convolutional layers due to the spatial structure being preserved. Each PyTorch module is then dispatched to a per-set-type operation. Piecewise-linear activations (ReLU, LeakyReLU, Sign) admit exact reachability via case-splitting and approximate reachability via relaxation, while smooth activations (Sigmoid and Tanh) use a shared S-curve relaxation. Intermediate-bound tightening and output-property evaluation are routed through a unified LP interface backed by HiGHS.

Frontends. The falsifier runs uniform random sampling and PGD with multiple restarts directly on the PyTorch model via autograd, returning a witness whenever one is found. The verifier consumes the reachable set(s) produced by the core and checks them against the property, returning `SAT` (with a witness), `UNSAT`, or `UNKNOWN`. For the competition, we compose the verifier and falsifier according to Section 2 and format the verdicts in the VNN-COMP convention.

4 Tool Setup and Configuration

n2v is open-source and available at https://github.com/sammsaski/n2v, with documentation at https://sammsaski.github.io/n2v/. The tool is installed by:

```
$ git clone --recurse-submodules <repo-url> n2v && cd n2v
$ pip install -r requirements.txt
$ pip install -e third_party/onnx2torch
$ pip install -e .[highs] # [highs] enables batched HiGHS backend
```

[2] `Reshape`, `Concat`, `Slice`, `Split`, `MatMul`, `Resize`, binary arithmetic, and reductions.

In contrast to NNV, no MATLAB license or proprietary toolboxes are required, and installation completes quickly on a clean Python environment. The equivalent MATLAB setup would require acquiring a MATLAB license, installing the MATLAB application itself, installing the five required toolboxes (Computer Vision, Deep Learning, Optimization, Parallel Computing, Statistics), and configuring the license before NNV could be installed.

A typical n2v workflow loads a model, constructs an input set, computes the output reachable set, and verifies the specification against it:

```python
import n2v
from n2v.sets import Star
from n2v.utils import load_vnnlib
from n2v.utils.verify_specification import verify_specification

model = n2v.utils.load_onnx("network.onnx")

# n2v's internal NN representation is a native PyTorch nn.Module
# the same object is used for inference, autograd, and reachability.
net = n2v.NeuralNetwork(model)

# Load the VNN-LIB specification (input bounds + unsafe region).
spec = load_vnnlib("spec.vnnlib")
input_star = Star.from_bounds(spec["lb"], spec["ub"])

# Compute the output reachable set.
output_sets = net.reach(input_star, method="approx")

# Verify the specification against the output reachable set(s).
# Returns 1 (unsat / safe), 0 (sat / counterexample), 2 (unknown).
result = verify_specification(output_sets, spec["prop"])
```

In the example above, the `method` keyword on `net.reach` selects between sound over-approximation (`"approx"`) and exact case-splitting (`"exact"`). Other keyword arguments tune reachability behavior, including `relax_factor` for relaxation strength, `lp_solver` for the LP backend, and `n_workers` for the number of parallel LP threads. Falsification (PGD or random sampling) is invoked separately.

5 Software Project and Contributors

n2v is an open-source MIT-licensed project, hosted on GitHub and maintained by the VeriVITAL research group at Vanderbilt University, where the main developer is Samuel Sasaki. n2v is built on more than a decade of work on NNV by Hoang-Dung Tran (original author and developer of NNV), Taylor T. Johnson, Diego Manzanas Lopez, Neelanjana Pal, Xiaodong Yang, Sung Woo Choi, Patrick Musau, Anne Tumlin, Serena Serbinowska, Luan Viet Nguyen, Mykhailo Ivashchenko, Stanley Bak, Weiming Xiang, Kerianne Hobbs, Feiyang Cai, Xenofon Koutsoukos, Tomoya Yamaguchi, Bardh Hoxha, and Danil Prokhorov, with

additional contributions to n2v from Ben Wooding and Hanchen David Wang. Published algorithms and case studies from this group underpin n2v's set representations, reachability, and verification methods.

The translation from MATLAB NNV to Python was carried out test-first, supported by an extensive test suite of three complementary kinds. Per-layer unit tests, over a thousand in total, cover each set-layer combination against analytical reachable sets on small examples. Approximately two hundred soundness tests verify that exact reachable sets are contained in their approximate counterparts and that forward passes of sampled inputs lie inside the computed output set. Differential tests compare n2v's reachable sets and verdicts against MATLAB NNV on shared benchmarks, providing a reference cross-check on the translation.

Acknowledgements. The material presented in this paper is based upon work supported by the National Science Foundation (NSF) through grant numbers 2220401 and 2443803 and the Defense Advanced Research Projects Agency (DARPA) under contract number FA8750-23-C-0518. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of DARPA or NSF.

Data-Availability Statement. The n2v source code is publicly available at https://github.com/sammsaski/n2v. Documentation and tutorials are available at https://sammsaski.github.io/n2v/. A Zenodo archive of the artifact corresponding to the VNN-COMP 2026 submission will be deposited following the competition, with a permanent DOI provided in the competition results.

References

1. Demarchi, S., Guidotti, D., Pulina, L., Tacchella, A.: Supporting standardization of neural networks verification with vnnlib and coconet. In: Narodytska, N., Amir, G., Katz, G., Isac, O. (eds.) Proceedings of the 6th Workshop on Formal Methods for ML-Enabled Autonomous Systems. Kalpa Publications in Computing, vol. 16, pp. 47–58. EasyChair (2023). https://doi.org/10.29007/5pdh, /publications/paper/Qgdn
2. Girard, A.: Reachability of uncertain linear systems using zonotopes. In: Morari, M., Thiele, L. (eds.) Hybrid Systems: Computation and Control. pp. 291–305. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
3. Hashemi, N., Sasaki, S., Oguz, I., Ma, M., Johnson, T.T.: Scaling Data-Driven Probabilistic Robustness Analysis for Semantic Segmentation Neural Networks. In: The Thirty-ninth Annual Conference on Neural Information Processing Systems (2025), https://openreview.net/forum?id=liefJOFVfH
4. Huangfu, Q., Hall, J.A.J.: Parallelizing the dual revised simplex method. Mathematical Programming Computation **10**(1), 119–142 (Mar 2018). https://doi.org/10.1007/s12532-017-0130-5, https://doi.org/10.1007/s12532-017-0130-5
5. Kaulen, K., Ladner, T., Bak, S., Brix, C., Duong, H., Flinkow, T., Johnson, T.T., Koller, L., Manino, E., Nguyen, T.H., Wu, H.: The 6th international verification of neural networks competition (vnn-comp 2025): Summary and results (2025), https://arxiv.org/abs/2512.19007

6. Lopez, D.M., Choi, S.W., Tran, H.D., Johnson, T.T.: NNV 2.0: The Neural Network Verification Tool. In: Computer Aided Verification: 35th International Conference, CAV 2023, Paris, France, July 17–22, 2023, Proceedings, Part II. p. 397–412. Springer-Verlag, Berlin, Heidelberg (2023). https://doi.org/10.1007/978-3-031-37703-7_19, https://doi.org/10.1007/978-3-031-37703-7_19
7. Roy, A., Antony, A., Gimelli, A., Daggitt, M.L.: VNN-LIB 2.0: Rigorous Foundations for Neural Network Verification (2026), https://arxiv.org/abs/2605.07451
8. Singh, G., Gehr, T., Püschel, M., Vechev, M.: An abstract domain for certifying neural networks. Proc. ACM Program. Lang. **3**(POPL) (Jan 2019). https://doi.org/10.1145/3290354, https://doi.org/10.1145/3290354
9. Tran, H.D., Bak, S., Xiang, W., Johnson, T.T.: Verification of deep convolutional neural networks using imagestars. In: Lahiri, S.K., Wang, C. (eds.) Computer Aided Verification. pp. 18–42. Springer International Publishing, Cham (2020)
10. 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: Formal Methods – The Next 30 Years: Third World Congress, FM 2019, Porto, Portugal, October 7–11, 2019, Proceedings. p. 670–686. Springer-Verlag, Berlin, Heidelberg (2019). https://doi.org/10.1007/978-3-030-30942-8_39, https://doi.org/10.1007/978-3-030-30942-8_39
11. Tran, H.D., Pal, N., Lopez, D.M., Musau, P., Yang, X., Nguyen, L.V., Xiang, W., Bak, S., Johnson, T.T.: Verification of piecewise deep neural networks: a star set approach with zonotope pre-filter. Formal Aspects of Computing **33**(4), 519–545 (Aug 2021). https://doi.org/10.1007/s00165-021-00553-4, https://doi.org/10.1007/s00165-021-00553-4
12. Tran, H.D., Pal, N., Musau, P., Lopez, D.M., Hamilton, N., Yang, X., Bak, S., Johnson, T.T.: Robustness verification of semantic segmentation neural networks using relaxed reachability. In: Silva, A., Leino, K.R.M. (eds.) Computer Aided Verification. pp. 263–286. Springer International Publishing, Cham (2021)
13. 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: Computer Aided Verification: 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21–24, 2020, Proceedings, Part I. p. 3–17. Springer-Verlag, Berlin, Heidelberg (2020). https://doi.org/10.1007/978-3-030-53288-8_1, https://doi.org/10.1007/978-3-030-53288-8_1
14. 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 verification. Advances in Neural Information Processing Systems **34** (2021)
15. Wu, H., Isac, O., Zeljić, A., Tagomori, T., Daggitt, M., Kokke, W., Refaeli, I., Amir, G., Julian, K., Bassan, S., Huang, P., Lahav, O., Wu, M., Zhang, M., Komendantskaya, E., Katz, G., Barrett, C.: Marabou 2.0: A versatile formal analyzer of neural networks. In: Gurfinkel, A., Ganesh, V. (eds.) Computer Aided Verification. pp. 249–264. Springer Nature Switzerland, Cham (2024)
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** (2020)
17. Xu, K., Zhang, H., Wang, S., Wang, Y., Jana, S., Lin, X., Hsieh, C.J.: Fast and Complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In: International Conference on Learning Representations (2021), https://openreview.net/forum?id=nVZtXBI6LNn