arXiv:2611.24114v1 [cs.LG] 18 Nov 2026

The 7th International Verification of Neural Networks Competition (VNN-COMP 2026): Summary and Results

Tobias Ladner*[1], Konstantin Kaulen*[2], Stanley Bak[3], Julien Girard-Satabin[4], Taylor T. Johnson[5], Edoardo Manino[6], ThanhVu H Nguyen[7], Ben Wooding[5], and Haoze Wu[8,9]

[1] Technical University of Munich, Munich, Germany
[2] RWTH Aachen University, Aachen, Germany
[3] Stony Brook University, Stony Brook, New York, USA
[4] Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France
[5] Vanderbilt University, Nashville, Tennessee, USA
[6] University of Manchester, Manchester, UK
[7] George Mason University, Fairfax, Virginia, USA
[8] Amherst College, Amherst, Massachusetts, USA
[9] VMware Research by Broadcom, USA

Abstract

This report summarizes the seventh International Verification of Neural Networks Competition (VNN-COMP 2026), held as part of the Symposium on AI Verification (SAIV) co-located with the International Conference on Computer-Aided Verification (CAV) at FLoC 2026 in Lisbon, Portugal. The goal of the competition is to provide an objective comparison of the state-of-the-art methods in neural network verification, in terms of scalability and speed. Along this line, we used standard formats (ONNX for neural networks and VNNLIB for specifications), standard hardware (all tools are run by the organizers on AWS), and tool parameters provided by the tool authors. This report summarizes the rules, benchmarks, participating tools, results, and lessons learned from this competition.

1 Introduction

Deep-learning-based systems are increasingly being deployed in a wide range of domains, including recommendation systems, computer vision, and autonomous driving. While the nominal performance of these methods has increased significantly over the last years, they largely lack formal guarantees on their behavior. However, in safety-critical applications, including autonomous systems, robotics, cybersecurity, and cyber-physical systems (CPS), such guarantees are essential for certification and reliability.

While the literature on the verification of traditionally designed systems is wide and successful, neural network verification remains an open problem, despite significant efforts over the last years. In 2020, the International Verification of Neural Networks Competition (VNN-COMP) was established to facilitate comparison between existing approaches, bring researchers working on this problem together, and help shape future directions of the field. VNN-COMP has been held annually since then [16, 97, 24, 23, 22, 72]. In 2026, the 7th iteration of the annual VNN-COMP[1] was held as a part of the third International Symposium on AI Verification (SAIV) [7] that was collocated with the 38th International Conference on Computer-Aided Verification (CAV) [8].

*Equal contribution.
[1] http://vnn-comp.github.io/

This 7th iteration of the VNN-COMP continues last year's trend of increasing standardization and automatization, aiming to enable a fair comparison between the participating tools and to simplify the evaluation of a large number of tools on a variety of (real-world) problems. As in the last iteration, VNN-COMP 2026 standardizes (1) neural network and specification formats, ONNX for neural networks and VNN-LIB [35] for specifications, (2) evaluation hardware, providing participants the choice of a range of cost-equivalent AWS instances with different trade-offs between CPU and GPU performance, and (3) evaluation pipelines, enforcing a uniform interface for the installation and evaluation of tools.

The competition was kicked off with the solicitation for participation in February 2026. Rule discussion started in March and rules were finalized in May 2026 (see an overview in Section 2). From April to June 2026, benchmarks were proposed and discussed. Meanwhile, the organizing team decided to continue using AWS as the evaluation platform and started to implement an automated submission and testing system for both benchmarks and tools. By mid-July 2026, eleven teams submitted their tools and the organizers evaluated all entrants to obtain the final results, discussed in Section 6 and presented at SAIV on July 24, 2026. Discussions were structured into three issues on the official GitHub repository[2]: rules discussion, benchmarks discussion, and tool submission. All submitted benchmarks[3] and final results[4] were aggregated in separate GitHub repositories.

The remainder of this report is organized as follows: Section 2 discusses the competition rules, Section 4 lists all participating tools, Section 5 lists all benchmarks, Section 6 summarizes the results, and Section 7 concludes the report, discussing potential future improvements.

[2] https://github.com/VNN-COMP/vnncomp2026/issues
[3] https://github.com/VNN-COMP/vnncomp2026_benchmarks
[4] https://github.com/VNN-COMP/vnncomp2026_results

2 Rules

Terminology An *instance* is defined by a property specification (pre- and post-condition), a network, and a timeout. For example, one instance might consist of an MNIST classifier with one input image, a given local robustness threshold ϵ, and a specific timeout. A *benchmark* is defined as a set of related instances. For example, one benchmark might consist of a specific MNIST classifier with 100 input images, potentially different robustness thresholds ϵ, and one timeout per input.

Run-time caps Run-times are capped on a per-instance basis, i.e., any verification instance will timeout (and be terminated) after at most X seconds, determined by the benchmark proposer. These can be different for each instance. The total per-benchmark runtime (sum of all per-instance timeouts) may not exceed 6 hours per benchmark. For example, a benchmark proposal could have six instances with a one-hour timeout, or 100 instances with a 3.6-minute timeout, each. To enable a fair comparison, we measure the startup overhead for each tool by running it on a range of tiny networks and subtract the minimal overhead from the total runtime.

Hardware To allow for comparability of results, all tools were evaluated on equal-cost hardware using Amazon Web Services (AWS). Each team could decide between a range of AWS instance types (see Table 1) providing a CPU, GPU, or mixed focus.

Table 1: Available AWS instances.

	vCPUs	RAM [GB]	GPU
p3.2xlarge	8	61	V100 GPU with 16 GB memory
m5.16xlarge	64	256	✗
g5.8xlarge	32	128	A10G GPU with 24 GB memory

Scoring The final score is aggregate as the sum of all benchmark scores. Each benchmark score is the number of points (sum of instance scores discussed below) achieved by a given tool, normalized by the maximum number of points achieved by any tool on that benchmark. Thus, the tool with the highest sum of instance scores for a benchmark will get a benchmark score of 100, ensuring that all benchmarks are weighted equally, regardless of the number of constituting instances.

Instance score Each instance is scored is as follows:

- Correct hold (property proven): 10 points;
- Correct violated (counterexample found): 10 points;
- Incorrect result: -150 points (penalty increased compared to 2022);
- Timeout / Runtime Error / Unknown: 0 points.

However, the ground truth for any given instance is generally not known a priori. In the case of disagreement between tools, we, therefore, place the burden of proof on the tool claiming that a specification is violated, i.e., that a counterexample can be found, and deem it correct exactly if it produces a valid counterexample.

The provided counterexamples were supposed to define both the input and the resulting output of the networks. However, for some tools and instances, the output definition was

either missing or differed from the network output as computed by the onnxruntime package used to evaluate counterexamples (by performing inference given the inputs). The competition rules were ambiguous how this would be handled. We decided to discard all outputs in the counterexample files and base the evaluation solely on the given inputs and their respective outputs as computed by the onnxruntime.

Time bonus As in the previous year, no time bonus was awarded. Instead, all tools that compute the correct result within the time limit receive the same amount of points.

Overhead Correction The overhead of tools was measured, but only used to adapt the timeouts. It did not influence the scores, as no time bonus was awarded. To measure the tool-specific overhead, we created trivial network instances and included those in the measurements. We then observed the minimum verification time over all instances and considered that to be the overhead time for the tool.

Format As since 2021, we standardized neural networks to be in `onnx` format, specifications in `vnnlib` format, and counterexamples in a format similar to the `vnnlib` format. Further, tool authors were required to provide scripts fully automating the installation process of their tool, including the acquisition of any licenses that might be needed. Similar to the previous year, a preparation and execution script had to be provided for running their tool on a specific instance consisting of a network file, specification file, and timeout. The specifications are interpreted as definitions of counterexamples, meaning that a property is proven "correct" if the specification is shown to be unsatisfiable, conversely, the property is shown to be violated if a counterexample fulfilling the specification is found. Specifications consisted of disjunctions over conjunctions in both pre- and post-conditions, allowing a wide range of properties from adversarial robustness over multiple hyper-boxes to safety constraints to be encoded. For example, robustness with respect to inputs in a hyper-box had to be encoded as disjunctive property, where any of the other classes is predicted.

Tracks Since VNN-COMP'24, the competition is split into two tracks, both of them scored: A "regular" track, and an "extended" track. The regular track consists of benchmarks selected by the tool participants based on a voting process, where a benchmark is included in the regular track if at least 50% of tool participants voted for it to be scored. Benchmarks with at least 1 vote and not included in the regular track were scored in the extended track. All benchmarks received at least one vote, so no benchmark was unscored.

3 Changes compared to previous edition

This section list the main differences in the competition organization compared to the previous year.

3.1 Support for a new benchmark format

Since the second edition of VNN-COMP, the VNN-LIB format [35] was chosen as a standard input format for the competition. VNN-LIB 1.0 was a subset of the SMT-LIB [19] language, widely used in Satisfaction Modulo Theory (SMT) solvers. Its simplicity allowed easy parsing by each prover.

However, VNN-LIB 1.0 lacked a well-defined syntax and a standardised parser, which resulted in each too having to re-implement their own parser and subtle differences in the syntax accepted by each tool. Furthermore, VNN-LIB 1.0 was insufficiently expressive [32], lacking for instance the ability to specify neural network types, properties involving linear arithmetic constraints, properties involving multiple applications of the same network or multiple applications of a single network.

In order to resolve these problems, VNN-LIB 2.0 was developed and released this year [101]. Compared to its predecessor, VNN-LIB 2.0 has a formal grammar, rigorous typing, handles nonlinear operations and multiple network declarations. Version 2.0 of the standard was accompanied by parsers for various programming languages, including C++, Python, Julia and Ocaml. Benchmark proposers were keen to adopt this new version, with all benchmarks in the regular track providing a VNN-LIB 2.0 version of their specification. Some benchmarks in the extended version provided VNN-LIB 2.0-only versions. Tools developers were able to handle most of VNN-LIB 2.0 benchmarks, and fell back to the 1.0 version whenever unknown constructs were met.

The aim next year is to move fully to VNN-LIB 2.0 as well as simplifying the competition infrastructure by using the standardised command-line format provided by the standard.

3.2 Hardened counter-example validation

For VNN-COMP 2026, the validation of counterexamples returned by participating tools was revised to address ambiguities arising from numerical tolerances. Counterexample validation is inherently challenging because verification specifications describe precise constraints, while both verification tools and the reference execution of ONNX networks rely on finite-precision floating-point arithmetic. Previous editions accommodated such numerical discrepancies by allowing violations of the specification constraints within a fixed tolerance. However, applying a tolerance directly to the feasible region changes the verification problem itself: an instance that is `unsat` under the original specification may become `sat` after its constraints are relaxed. Consequently, two contradictory answers could in principle both be considered correct, with one verifier proving the original property and another returning a counterexample that only satisfies the tolerance-relaxed specification.

For VNN-COMP 2026, submitted counterexamples were therefore validated against the original input specification without tolerance. The submitted input was evaluated in a fixed reference environment using ONNX Runtime 1.26.0 and ONNX 1.21.0 under Python 3.12 on CPU of the tool author's chosen evaluation hardware, and the resulting outputs were checked against the original VNN-LIB output constraints. A counterexample was classified as `CORRECT` only if the input and the reference network output satisfy the specification without tolerance,

and only such counterexamples received the regular score for a correct `sat` result. At the same time, requiring exact agreement on the output constraints would introduce a different problem: a mathematically valid counterexample close to the decision boundary may be evaluated slightly differently by a verification tool and by the reference ONNX Runtime execution due to floating-point arithmetic, different implementations of network operations, or different evaluation orders. Counterexamples whose inputs satisfy the specification exactly and whose reference outputs satisfy the output constraints only up to an absolute tolerance of 10^{-4} are therefore classified separately as `CORRECT_WITH_TOLERANCE`. These results received full points (*i.e.*, were not penalised as `INCORRECT`), but did not establish a correct `sat` result that could be used to penalise `unsat` results given by other tools.

This rule does not completely eliminate the fundamental numerical challenges of counterexample validation. In particular, the output of the fixed reference implementation is itself a finite-precision computation and is not necessarily identical to the real-arithmetic semantics implicitly assumed by many verification algorithms. Counterexamples located extremely close to a property boundary may consequently still change classification depending on numerical details of network execution. Fixing the ONNX Runtime version, execution platform, and numerical tolerance makes this behavior reproducible for the competition, but cannot provide an implementation-independent notion of validity for such borderline cases. The revised rule therefore represents a compromise: numerical tolerances are prevented from enlarging the input domain or turning an originally `unsat` problem into a valid `sat` result, while small numerical discrepancies in network evaluation are not penalised.

3.3 Open-sourcing the evaluation platform

We published the evaluation platform: https://github.com/VNN-COMP/vnn-eval-platform.

With more tool authors participating in ever more benchmarks, this comes with a high financial cost to run all submissions on AWS instances. Open-sourcing the evaluation platform enables anyone wanting to participate to set up their own instance of the VNN-COMP evaluation platform on their internal lab server. Any submission to such a local instance then mimics an execution on an AWS instance by running the same benchmark orchestration within a Docker container, thus enabling thorough testing of any tool ahead of the actual submission without stressing our budget. In the future, this also makes it easier for us to move off running VNN-COMP on AWS.

We are aware that this assumes that anyone who wants to participate has access to some server they can utilize for this; let us know if this is not the case, and, more broadly, do not hesitate to open an issue on the GitHub repository in case you encounter any bugs, have feature requests, or want to share other thoughts!

4 Participants

We list the tools and teams that participated in the VNN-COMP 2026 in Table 2 and reproduce their own descriptions of their tools below.

Table 2: Summary of the key features of participating tools. The hardware column describes the used AWS instance with `p3` and `g5` making GPUs available, see Table 1 for more details. Licenses refer to the external licenses required to use the corresponding tool, not the licensing of the tool itself.

Tool	References	Organizations	Place	Hardware	Licenses
α,β-CROWN	[131, 132, 126, 137, 105]	UIUC, UCLA, Drexel, Duke, RWTH Aachen	**1**	g5	GUROBI
ACT	[135, 53, 100, 90, 136]	University of New South Wales	10	g5	GUROBI (optional)
Agate	-	None	13	g5	-
Bright	[111, 132, 126]	University of Kent	7	g5	-
CORA	[10, 79, 80, 78, 84]	Technical University of Munich	6	g5	MATLAB
NeuralSAT	[43, 48, 45]	George Mason University	4	g5	GUROBI
nnenum	[17, 14]	Stony Brook University	11	m5	-
NNV	[121, 91, 123]	Vanderbilt University	5	g5	MATLAB
n2v	[103]	Vanderbilt University	8	g5	-
PyRAT	[87, 85]	Université Paris-Saclay, CEA-List	3	g5	closed source CEA license
SobolBox	[5]	IIT Gandhinagar	12	m5	-
SuperSimple	-	RWTH Aachen	9	m5	-
Vibecheck	[15, 18]	Stony Brook University	2	g5	-

4.1 α,β-CROWN

Team Co-leaders: Xiangru Zhong (student lead) and Huan Zhang (faculty advisor); Team members: Duo Zhou, Lei Huang, Yifei Li, Jingyi Wang, Xuanle Li, Ashmit Dutta, Lingyin Zhang. All members are from the University of Illinois Urbana-Champaign (UIUC).

The team **acknowledges** Hesun Chen, Haoyu Li, Ruize Gao, Hao Cheng, and Taobo Liao, who were involved in the development of the verifier in the year of 2026 but did not directly work on any benchmarks of the competition. A full list of contributors is available at `https://abcrown.org`.

Description α,β-CROWN (`alpha-beta-CROWN`) is an efficient neural network verifier based on the linear bound propagation framework and built on a series of works on bound-propagation-based neural network verifiers: CROWN [138], auto_LiRPA [131], α-CROWN [132], β-CROWN [126], GCP-CROWN [137], GenBaB [105], BICCOS [141]. The core techniques in α,β-CROWN combine the efficient and GPU-accelerated linear bound propagation method with branch-and-bound methods specialized for neural network verification.

The linear bound propagation algorithms in α,β-CROWN are based on our `auto_LiRPA` library [131], which supports general neural network architectures (including convolutional layers, pooling layers, residual connections, recurrent neural networks, Transformers, and general computation graphs) and a wide range of nonlinear functions (e.g., ReLU, tanh, trigonometric functions, sigmoid, max pooling and average pooling), and is efficiently implemented on GPUs with Pytorch and CUDA. We jointly optimize intermediate layer bounds and final layer bounds

using gradient ascent (referred to as α-CROWN or optimized CROWN/LiRPA [132]). Most importantly, we use branch and bound [25] (BaB) and incorporate split constraints in BaB into the bound propagation procedure efficiently via the β-CROWN algorithm [126], use cutting-plane method in GCP-CROWN [137] and BICCOS [141] to further tighten the bound, and support general nonlinearities in the branch-and-bound by GenBaB [105]. For smaller networks, we also use a mixed integer programming (MIP) formulation [114] combined with tight intermediate layer bounds from α-CROWN (referred to as α-CROWN + MIP [137]). The combination of efficient, optimizable and GPU-accelerated bound propagation with BaB produces a powerful and scalable neural network verifier.

Link https://github.com/Verified-Intelligence/alpha-beta-CROWN (main version)

Competition submission https://github.com/Verified-Intelligence/alpha-beta-CROWN_vnncomp2026 (only for reproducing competition results; please use the main version for other purposes)

Hardware and licenses CPU and GPU with 32-bit or 64-bit floating point; Gurobi license required for certain benchmarks.

Participated benchmarks All benchmarks.

4.2 ACT

Team Guanqin Zhang, Yulei Sui (University of New South Wales, Sydney).

Description ACT (Abstract Constraint Transformation) is an open-source framework for neural network testing and verification, built around a three-tier architecture comprising a front-end, back-end, and pipeline, with native PyTorch support.

The front-end supports model and specification synthesis and translation, converting diverse models and specifications into ACT's intermediate representation (IR). The back-end performs abstract execution and tensor-based branch-and-bound refinement, with constraint generation decoupled from constraint solving. Its search design draws on several prior works, including eager falsification [56], stochastic optimization-based falsification [135], MCTS-based exploration [53], and verdict boundary mining [100], among others. ACT's back-end also checks consistency between sampled concrete executions and the abstract bounds produced by abstract execution to detect unsoundness in verifier implementations [90].

Finally, the ACT pipeline connects the front-end and back-end and provides tensor-based testing and verification capabilities, including fuzzing [136]. This enables parallel mutation, execution, and property checking across different test inputs, while tensor-based refinement of verification subproblems exploits CPU/GPU parallelism to scale branch-and-bound-based verification.

Link https://github.com/SVF-tools/ACT

Commit 492d904d5bc2bb09438a538089f8381d0d75fe4e

Hardware and licenses CPU and GPU; Gurobi license (optional).

Participated Benchmarks `acasxu_2023`, `cersyve`, `challenging_certified_training_2026`, `cifar100_2024`, `cora_2024`, `dist_shift_2023`, `malbeware`, `relusplitter_2026`, `safenlp_2024`, `sat_relu`, `tinyimagenet_2024`, `tllverifybench_2023`, `vggnet16_2022`, `vit_2023`.

4.3 Agate

Team David Shriver (unaffiliated).

Description Agate is an early implementation of a neural network verification framework. It is still a work in progress. Agate takes in an ONNX model and behavior specification (such as a VNNLIB 2.0 counter example specification) and transforms them into a single model that expresses the satisfiability of the problem. This transformation is similar to the one used by DNNF [109], but combines all subproblems into a single model. The initial implementation submitted encompassed a simple adversarial attack solver and basic interval bound propagation with input branch and bound for complete verification. These approaches were run in parallel and both used the GPU, except for very small models. The only benchmark specific tuning done was to reduce the batch size for benchmarks with large models. The backend was implemented in Jax and supported most common ONNX operators, but notably, could not support operations with input dependent output shapes, such as dynamic slicing, at the time of submission. It uses a custom VNNLIB parser that did not support strict inequality at the time of submission. Finally, a last minute bug in the counter-example post-processor resulted in invalid witnesses, contributing to our poor competition performance. This bug has since been fixed, and our test suite is significantly improved to detect such regressions in the future.

Link https://github.com/veridnn/agate-vnncomp2026

Commit a1227f8c46e215e723de35a620e2105a8b4f83aa

Hardware and licenses CPU and GPU. Full source available on request.

Participated Benchmarks `acasxu_2023`, `cersyve`, `cgan2026`, `challenging_certified_training_2026`, `collins_rul_cnn_2022`, `cora_2024`, `dist_shift_2023`, `linearizenn_2024`, `lsnc_relu`, `malbeware`, `metaroom_2023`, `ml4acopf_2024`, `nn4sys`, `relusplitter_2026`, `safenlp_2024`, `sat_relu`, `tllverifybench_2023`.

4.4 Bright

Team Max Peeperkorn and Elena Botoeva (University of Kent)

Description Bright is a composable branch-and-bound neural network verifier written in JAX. It implements state-of-the-art GPU-accelerated bound propagation techniques [111, 132, 126] and is designed as a versatile research tool that emphasises customisation to easily integrate custom branching heuristics and procedures to offload domains to MIP under certain conditions. The tool is still under development and currently supports fully-connected and convolutional networks with ReLU activations.

Links https://git.cs.kent.ac.uk/bright/bright-vnncomp-2026

Commit b2351384b23233b440ee35fd64dc6e3d54d08da6

Hardware and licenses CPU and GPU, Gurobi Licence

Participated benchmarks acasxu_2023, cifar100_2024, cora_2024, linerizenn_2024, malbeware, metaroom_2023, relusplitter_2026, safenlp_2024, sat_relu, tinyimagenet_2024, tllverifybench_2023

4.5 CORA

Team Lukas Koller, Tobias Ladner, Benedikt Kellner, Matthias Althoff (Technical University of Munich).

Description CORA [10, 11] enables the formal verification of neural networks, both in open-loop as well as in closed-loop scenarios [78, 84]. Open-loop verification refers to the task where properties of the output set of a neural network are verified, e.g., correctly classified images given noisy input, as also considered at VNN-COMP. In closed-loop scenarios, the neural network is used as a controller of a dynamic system, e.g., controlling a car while keeping a safe distance

over some time horizon. Moreover, CORA can also train robust neural networks [79, 129], which uses an efficient batch-wise propagation of zonotopes through a neural network on a GPU.

For the competition, reachability analysis with zonotopes is used to compute a convex enclosure of the output set of a neural network. In addition to simultaneous input and neuron splitting, a specification-driven input refinement procedure iteratively encloses the set of unsafe inputs [80]. All operations use batch-wise computations to take full advantage of GPU-acceleration. Further details on our approach can be found in our competition contribution paper [81].

Link https://github.com/toladnertum/vnncomp-cora-2026

Commit 798551ab7e7cba1c2c8a3a14726478cbfc8690e6

Hardware and licenses GPU, MATLAB license.

Participated Benchmarks `acasxu`, `cersyve`, `cgan2026`, `challenging_certified_training_2026`, `cifar100_2024`, `collins_rul_cnn_2022`, `cora_2024`, `dist_shift_2023`, `linearizenn_2024`, `malbeware`, `metaroom_2023`, `nn4sys`, `safenlp_2024`, `sat_relu`, `soundnessbench_2026`, `tinyimagenet_2024`, `tllverifybench_2023`, `isomorphic_acasxu_2026`, `monotonic_acasxu_2026`.

4.6 NeuralSAT

Team Hai Duong and Thanhvu Nguyen (George Mason).

Description NeuralSAT [43, 48, 45, 42, 46, 44, 38, 41, 47, 39, 40] integrates conflict-driven clause learning (CDCL) in SAT/SMT-solving with an DNN abstraction-based theory solver for infeasibility checking. NeuralSAT implements the DPLL(T) framework used in modern SMT solvers such as Z3. The design of NeuralSAT is inspired by the core algorithms used in SMT solvers such as CDCL components and theory solving. The tool is written in Python and uses Gurobi for LP solving. Unlike many other modern NNV tools, NeuralSAT *does not* require parameter tuning and works out of the box, e.g., the tool runs on the wide-range of benchmarks in VNN-COMPs without any tuning.

Link https://github.com/dynaroars/neuralsat

Commit 8c6a493d2b9314b06c6f19cd452f1c7ab5bd2657

Hardware and licences GPU, Gurobi License

Participated benchmarks All benchmarks in the regular track.

4.7 nnenum

Team Doug Wehbe, Eli Pregerson, Stanley Bak (Stony Brook University)

Description The nnenum tool [14] uses multiple levels of abstraction to achieve high-performance verification of ReLU networks without sacrificing completeness [13]. The core verification method is based on reachability analysis using star sets [119], combined with the ImageStar method [116] to propagate sets through all linear layers supported by the ONNX runtime, such as convolutional layers with arbitrary parameters. The tool is written in Python 3 and uses GLPK for LP solving. This year we improved handling of sparse transformations like convolution layers to use less memory, as well as added some support for residual connections.

Link https://github.com/wehbedoug/nnenum-2026

Commit fa2a14d736973bfb728856afaffa35ed6cfae0ef

Hardware and licences CPU, Gurobi license (optional)

Participated benchmarks `acasxu`, `cgan`, `collins-rul-cnn`, `cora`, `linearizenn`, `metaroom`, `nn4sys`, `relusplitter`, `safeNLP`, `sat-relu`, `tllverifybench`, `vggnet16`.

4.8 NNV

Team Samuel Sasaki, Anne M. Tumlin, Ben Wooding, Taylor T. Johnson (Vanderbilt University)

Description The Neural Network Verification (NNV) Tool [121, 91, 92, 123] is a formal verification software tool for deep learning models and cyber-physical systems with neural network components written in MATLAB and available at https://github.com/verivital/nnv. NNV uses a star-set state-space representation and reachability algorithm that allows for a layer-by-layer computation of exact or overapproximate reachable sets for feed-forward [119], convolutional [116], semantic segmentation (SSNN) [120], and recurrent (RNN)[118] neural networks, as well as neural network control systems (NNCS) [117, 121] and neural ordinary differential equations (Neural ODEs) [95]. NNV3, the latest version [123], considers weight perturbations [143, 144], graph neural networks (GNNs) [124], videos [102], and fairness [122]. The star-set based algorithm is naturally parallelizable, which allows NNV to be designed to perform efficiently on multi-core platforms. Additionally, if a particular safety property is violated, NNV can be used to construct and visualize the complete set of counterexample inputs for a neural network (exact-analysis). For this competition, updated from last year's, we tailor the solver approach depending on the benchmark at hand, although all follow a similar flow. First, we perform a simulation-guided search for counterexamples for a fixed number of samples. If no counterexamples are found (i.e., demonstrate that the property is SAT), then we utilize an iterative refinement approach using reachability analysis to verify the property (UNSAT). This consists of performing reachability analysis using a relax-approximation method [120], if not verified, then a less conservative approximation based on zonotope pre-filtering approach [115], and finally using the exact analysis when possible [116] until the specification is verified or there is a timeout. Based on the benchmark to evaluate, the initial reachability analysis may be any of the overapproximation methods or the exact method, based on the complexity of the benchmarks (size of network, input, etc). In addition to the star set-based approaches, we implement another approach based on conformal prediction that can scale better to larger networks and input sizes, although only probabilistic guarantees can be provided [57]. This is the only method that uses GPU, all others run solely on CPU.

Link https://github.com/verivital/nnv

Commit 376568b449535a779f751dd2def774a0ed94ead8

Hardware and licenses CPU and GPU, MATLAB license.

Participated Benchmarks Regular track (all) and Extended track (vggnet16, ml4acopf, reluspliter).

4.9 n2v

Team Samuel Sasaki, Ben Wooding, Anne M. Tumlin, Taylor T. Johnson (Vanderbilt University)

Description n2v [103] is an open-source Python toolbox for neural network verification, entering the competition this year. n2v is a successor to the MATLAB-based NNV tool [121, 91, 123], reimplemented in Python with native PyTorch integration to lower the barrier to adoption for the deep learning and verification communities. n2v supports exact and over-approximate reachability using star sets, zonotopes, and hyperrectangles. In addition, n2v contributes probabilis-

tic verification via conformal inference, PGD-based falsification, and a batched HiGHS-backed linear-programming pipeline.

Link https://github.com/sammsaski/n2v

Commit 2b9ddcb39ca61780353e4416fd34d9d181538478

Hardware and licenses CPU and GPU, fully open-source.

Participated Benchmarks All benchmarks.

4.10 PyRAT

Team Augustin Lemesle, Sasha Cuau, Serge Durand, Tristan Le Gall (CEA-List), Julien Lehmann (OCamlPro)

Description PyRAT (Python Reachability Assessment Tool) [87, 86] is an abstract interpretation based tool to verify the safety and robustness of neural networks. PyRAT relies on multiple zonotopic domains to efficiently compute the reachable states for various architectures of neural networks (fully-connected, convolutionnal, recurrent, transformers, ...). It supports multiple non-linear activation functions (ReLU, Sigmoid, Softmax, Floor, ...) with precise abstractions while maintaining correctness and floating-point soundness on CPU. PyRAT can also leverage GPU-parallelization to scale to large networks or to batch analyses. For smaller networks and problems, PyRAT can leverage branch and bound strategies on the inputs with heuristics like ReCIPH [49]. While on larger ReLU networks, PyRAT will use branch and bound strategies on the ReLU neurons in conjunction with fast GPU computation.

For this year competition, PyRAT extended its property support to VNN-LIB 2.0 supporting its new paradigms like multiple inputs, networks and more complex properties. Relying on our relational zonotopic domains allowed PyRAT to address these new and complex relations between inputs and networks.

Link https://git.frama-c.com/pub/pyrat

Commit 2dc848d64f987e283b6b85d739c6f5d6dfc6dd91

Hardware and licenses CPU and GPU, closed source CEA licence (free academic licence available for research purposes).

Participated benchmarks All benchmarks.

4.11 SobolBox

Team Sarthak Das (Indian Institute of Technology Gandhinagar)

Description SobolBox is a spiritual successor to the unsubmitted INNVerS [4] project, developed by Shubhajit Roy (IIT Gandhinagar) and Avishek Lahiri (IACS Kolkata) for VNNCOMP-2021, under the supervision of Dr. Rajarshi Ray (IACS Kolkata).

SobolBox [5, 33] is a Python-based black-box falsification tool for detecting safety violations in neural networks. It accepts ONNX-format models and VNNLIB 1.0 safety specifications, treating neural networks as multi-input, multi-output, non-convex black boxes without making architectural assumptions. Input bounds are extracted from VNNLIB specifications using the Z3 theorem prover, after which low-discrepancy candidate points are generated using Sobol sequence sampling [112].

The core premise of SobolBox is that safety violations, if any, are most likely to occur near the optima. A preliminary pass over Sobol samples is used to quickly detect counterexamples before optimization, if they exist. For the remaining instances, promising regions are selected from the extrema of the sampled network outputs and refined using a combination of global

PSO [74] and local L-BFGS-B optimization [26]. Candidate optima are checked for safety violations, with Sign-OPT searches [30] additionally performed around the optima to detect nearby counterexamples.

The resulting output-bound under-approximations are checked against the safety property using Z3. SobolBox reports `sat` with a counterexample when a violation is found, `unsat` when the computed output bounds exclude a violation, or `unknown` when the analysis is inconclusive. Since the computed output bounds are under-approximations, `unsat` results are high-confidence but unsound. Networks whose effective input dimension exceeds the 21,201-dimensional limit of the employed Joe-Kuo Sobol construction [65] are reported as `unknown`. SobolBox also employs memoization of black-box evaluations, parallelization, and caching of Sobol samples and computed output bounds to reduce computational overhead across runs.

For benchmarks with binary inputs and outputs, SobolBox provides an alternate discrete-search mode using rounded Sobol samples and Nevergrad's `PortfolioDiscreteOnePlusOne` optimizer; this mode does not report `unsat` since discrete search being stochastic can't mathematically prove the property holds everywhere, even within the underapproximated output bounds.

Links https://codeberg.org/tech-absurdist/SobolBox

Commit 64826496ec80bdc9f9925aefd9789d880e94272b

Hardware and licenses CPU, no license required.

Participated benchmarks `acasxu`, `cersyve`, `cgan2026`, `collins_rul_cnn_2022`, `dist_shift_2023`, `linearizenn_2024`, `metaroom_2023`, `ml4acopf_2024`, `safenlp_2024`, `sat_relu`, `tllverifybench_2023`.

4.12 SuperSimple

Team Konstantin Kaulen (RWTH Aachen University)

Description SuperSimple is an intentionally minimal and well-documented neural network verification tool developed as an educational and competition baseline for VNN-COMP. It implements a deliberately simple verification pipeline: first, it searches for counterexamples using projected gradient descent (PGD); subsequently, it attempts to prove the property using incomplete bound propagation. For the latter, SuperSimple builds on `auto_LiRPA` [131] and successively applies CROWN [138] and optimized CROWN (α-CROWN) [132]. It returns `unsat` whenever the resulting bounds suffice to prove that the given property holds, and `unknown` otherwise.

This submission of a simple tool serves several purposes. First, in light of the increasing computational demands of complete verification [71] and the complexity of configuring state-of-the-art verification systems [70], SuperSimple provides a lightweight and easy-to-use baseline for assessing how many VNN-COMP instances can already be solved using comparatively simple techniques and modest computational resources. This provides an indication of the verification performance accessible to non-expert practitioners without highly specialized verification pipelines or large-scale compute environments. Second, SuperSimple can reveal benchmarks for which such baseline techniques are already sufficient and may, in some cases, even outperform more sophisticated verification systems in terms of running time. Such cases can highlight unnecessary algorithmic overhead, suboptimal configuration, or opportunities for improving the efficiency of state-of-the-art verifiers. Finally, SuperSimple helps identify benchmarks that have become too easy to meaningfully discriminate between modern verification approaches: if almost all instances can be solved using this simple pipeline, the benchmark may be regarded as largely solved and of limited value for measuring further progress at the state of the art.

Links https://github.com/kkaulen/simple_vnn_comp_verifier

Commit 780317d108e0de8aa9b29b4773079a9c4607213e

Hardware and licenses CPU or GPU, no license required.

Participated benchmarks All benchmarks (scored and unscored).

4.13 vibecheck

Team Doug Wehbe, Stanley Bak (Stony Brook University)

Description The vibecheck formal verification tool is a high-performance, vibe-coded decision procedure for neural networks. It was created using Claude Code, mostly Opus 4.8, over the course of two months. It combines a variety of analysis techniques: bounds propagation, zonotope analysis, star set abstract domain, MILP solving, α-CROWN-style slope optimization for ReLU relaxations, and a (maybe new) high performance GPU-based branch and bound solver to handle splitting. A blog post is available [15] detailing the development process and experience.

Links https://github.com/stanleybak/vibecheck-nn

Commit 341c56bd5ca719d0935b9b6659c8610158974197

Hardware and licenses GPU, Gurobi

Participated benchmarks All benchmarks (scored and unscored).

5 Benchmarks

In this section, we provide an overview of all scored benchmarks, reproducing the benchmark proposers' descriptions. Artifacts for all benchmarks are available in the repository[5]. Table 4 presents a summary of all benchmarks. Table 3 presents statistics across all benchmarks.

Table 3: Benchmark statistics per track. **VNNLIB** lists the specification-format versions each benchmark ships; **Budget** is the sum of the per-instance timeouts in hours.

Benchmark	VNNLIB	Networks	Specs	Instances	Budget (h)
Regular track					
acasxu_2023	1.0, 2.0	45	10	186	6.0
cersyve	1.0, 2.0	12	3	12	0.3
cgan2026	1.0, 2.0	8	21	21	6.0
challenging_certified_training_2026	1.0, 2.0	6	60	60	6.0
cifar100_2024	1.0, 2.0	2	200	200	5.6
collins_rul_cnn_2022	1.0, 2.0	3	41	62	6.0
cora_2024	1.0, 2.0	9	60	180	1.5
dist_shift_2023	1.0, 2.0	1	72	72	6.0
linearizenn_2024	1.0, 2.0	10	60	60	15.0
lsnc_relu	1.0, 2.0	1	80	80	0.6
malbeware	1.0, 2.0	3	50	150	4.2
metaroom_2023	1.0, 2.0	100	100	100	5.8
ml4acopf_2024	1.0, 2.0	9	23	69	11.5
nn4sys	1.0, 2.0	11	182	194	6.0
relusplitter_2026	1.0, 2.0	104	20	120	6.0
safenlp_2024	1.0, 2.0	1	1054	1080	6.0
sat_relu	1.0, 2.0	100	100	100	2.8
soundnessbench_2026	1.0, 2.0	2	60	60	2.5
tinyimagenet_2024	1.0, 2.0	1	200	200	5.6
tllverifybench_2023	1.0, 2.0	32	32	32	5.3
traffic_signs_recognition_2023	1.0, 2.0	3	45	45	6.0
vggnet16_2022	1.0, 2.0	1	18	18	6.0
vit_2023	1.0, 2.0	2	200	200	5.6
yolo_2023	1.0, 2.0	1	72	72	6.0
Regular total		**467**	**2763**	**3373**	**132.2**
Extended track					
adaptive_cruise_control_non_linear_2026	2.0	1	50	50	1.4
cctsdb_yolo_2023	1.0, 2.0	2	39	39	3.8
collins_aerospace_benchmark	1.0, 2.0	1	6	6	6.0
isomorphic_acasxu_2026	2.0	50	50	50	1.4
monotonic_acasxu_2026	2.0	31	50	50	1.4
smart_turn_multimodal_2026	2.0	1	50	50	1.4
Extended total		**86**	**245**	**245**	**15.3**
Overall		**553**	**3008**	**3618**	**147.5**

[5] https://github.com/VNN-COMP/vnncomp2026_benchmarks

Table 4: Overview of all scored benchmarks.

Category	Benchmark	Application	Network Types	# Params	Effective Input Dim	Track
Complex	cGAN	Image Generation & Image Prediction	Conv. + Vision Transformer	500k - 68M	5	regular
	NN4Sys	Learned Index, Cardinality Estimation & Adaptive Bitrate Selection	FC + Conv1D, ReLU + Sigmoid	33k - 37M	1-308	regular
	LinearizeNN	NN controller approximation	FC. + Conv. + Vision Transformer + Residual + ReLU	203k	4	regular
	ml4acopf	Power System	Complex (ReLU + Trigonometric + Sigmoid)	4k-680k	22 - 402	extended
	ViT	Vision	Conv. + Residual + Softmax + BatchNorm	68k - 76k	3072	extended
	Collins Aerospace	-	FC + Conv. + Residual, LeakyReLU + MaxPool + Square	1.8M	1.2M	extended
	LSNC-ReLU	Lyapunov stability of NN controllers	FC, ReLU	275	6	extended
	CCTSDB	-	FC + Conv. + Residual, ReLU + MaxPool + Clip	100k	2	extended
CNN & ResNet	Collins RUL CNN	Condition Based Maintenance	Conv. + ReLU, Dropout	60k - 262k	400 - 800	regular
	VGGNet16	Image Classification	Conv. + ReLU + MaxPool	138M	150k	extended
	Traffic Signs Recognition	Image Classification	Conv. + Sign + MakPool + BatchNorm	905k - 1.7M	2.7k - 12k	extended
	cifar100	Image Classification	FC + Conv. + Residual, ReLU + BatchNorm	2.5M - 3.8M	3072	regular
	tinyimagenet	Image Classification	FC + Conv. + Residual, ReLU + BatchNorm	3.6M	9408	regular
	Metaroom	-	Conv. + FC, ReLU	466k - 7.4M	5376	regular
	Yolo	-	FC + Conv. + Residual, ReLU + Sigmoid	22k - 37M	1 - 308	extended
	SoundnessBench	Verifier Test	FC + Conv. + ReLU	1.7M	12k	regular
	Relusplitter	Meta-Benchmark	FC + Conv. + ReLU	13k - 625k	5 - 3k	extended
	MalBeWare	Malware Classification	FC + Conv. + ReLU	102k - 1.5M	4k	regular
	cersyve	Neural Certificates for Control Systems	FC + Residual + ReLU	1k - 4k	2 - 6	regular
FC	TLL Verify Bench	Two-Level Lattice NN	Two-Level Lattice NN (FC. + ReLU)	17k - 67M	2	regular
	Acas XU	Collision Detection	FC. + ReLU	13k	5	regular
	Dist Shift	Distribution Shift Detection	FC. + ReLU + Sigmoid	342k - 855k	792	regular
	safeNLP	Sentence classification	FC. + ReLU	4k	30	regular
	CORA	Image Classification	FC. + ReLU	575k, 1.1M	784, 3072	regular
	SAT ReLU	Verifier Test	FC. + ReLU	100-35k	2 - 100	regular

5.1 cGAN

Proposed by Feiyang Cai, Ali Arjomandbigdeli, Stanley Bak (Stony Brook University)

Motivation While existing neural network verification benchmarks focus on discriminative models, the exploration of practical and widely used generative networks remains neglected in terms of robustness assessment. This benchmark introduces a set of image generation networks specifically designed for verifying the robustness of the generative networks.

Networks The generative networks are trained using conditional generative adversarial networks (cGAN), whose objective is to generate camera images that contain a vehicle obstacle located at a specific distance in front of the ego vehicle, where the distance is controlled by the input distance condition. The network to be verified is the concatenation of a generator and a discriminator. The generator takes two inputs: 1) a distance condition (1D scalar) and 2) a noise vector controlling the environment (4D vector). The output of the generator is the generated image. The discriminator takes the generated image as input and outputs two values: 1) a real/fake score (1D scalar) and 2) a predicted distance (1D scalar). Several different models with varying architectures (CNN and vision transformer) and image sizes (32x32, 64x64) are provided for different difficulty levels.

Specifications The verification task is to check whether the generated image aligns with the input distance condition, or in other words, verify whether the input distance condition matches the predicted distance of the generated image. In each specification, the inputs (condition distance and latent variables) are constrained in small ranges, and the output is the predicted distance with the same center as the condition distance but with slightly larger range.

Link https://github.com/feiyang-cai/cgan_benchmark2023

5.2 NN4Sys

Proposed by the α,β-CROWN team with collaborations with Shuyi Lin and Cheng Tan at Northeastern University.

Application The benchmark contains networks for database learned index, video streaming learned adaptive bitrate, and learned cardinality estimation which map inputs from various dimensions to 1-dimension outputs.

- *Background*: learned index, learned cardinality, and learned adaptive bitrate are all instances in neural networks for computer systems (NN4Sys), which are neural network based methods performing system operations. These classes of methods show great potential but have one drawback—the outputs of an NN4Sys model (a neural network) can be arbitrary, which may lead to unexpected issues in systems.

- *What to verify*: our benchmark provides multiple pairs of (1) trained NN4Sys model and (2) corresponding specifications. We design these pairs with different parameters such that they cover a variety of user needs and have varied difficulties for verifiers. We describe benchmark details in our NN4SysBench paper [89].

- *Translating NN4Sys applications to a VNN benchmark*: the original NN4Sys applications have some sophisticated structures that are hard to verify. We tailored the neural networks and their specifications to be suitable for VNN-COMP. For example, learned index [83] contains multiple NNs in a tree structure that together serve one purpose. However, this cascading structure is inconvenient/unsupported to verify because there is a "switch" operation—choosing one NN in the second stage based on the prediction of the first stage's NN. To convert learned indexes to a standard form, we train a monolithic (larger) NN.

- *A note on broader impact*: using NNs for systems is a broad topic, but many existing works lack strict safety guarantees. We believe that NN Verification can help system developers gain confidence to apply NNs to critical systems. We hope our benchmark can be an early step toward this vision.

Networks This benchmark has eleven networks with different parameters: two for learned indexes, four for learned cardinality estimation and five for learned adaptive bitrate. The learned index uses fully-connected feed-forward neural networks. The other two—the learned cardinality and the learned adaptive bitrate—have a relatively sophisticated internal structure. Please see our NN4SysBench paper [89] for details.

Specifications For learned indexes, the specification aims to check if the prediction error is bounded. The specification is a collection of pairs of input and output intervals such that any input in the input interval should be mapped to the corresponding output interval. For learned cardinality estimation and learned adaptive bitrate, the specifications check the prediction error bounds (similar to the learned indexes) and monotonicity of the networks. By monotonicity specifications, we mean that for two inputs, the network should produce a larger output for the larger input, which is required by cardinality estimation or adaptive bitrate.

Link: https://github.com/Khoury-srg/VNNComp26_NN4Sys

5.3 LinearizeNN

Proposed by Ali Arjomandbigdeli, Stanley Bak (Stony Brook University).

Motivation Assuming having a neural network controller approximation with a piecewise linear model in the form of a set of linear models with added noise to account for local linearization error. The objective of this benchmark is to investigate the neural network output falls within the range we obtain from our linear model output plus some uncertainty. The idea of this benchmark came from one of our recent paper [12] in which we approximated the NN controller with a piecewise linear model, and we wanted to check if the neural network output falls within the range we obtained from our linear model output plus some uncertainty.

Networks The neural network controller we used in this benchmark is an image-based controller for an Autonomous Aircraft Taxiing System whose goal is to control an aircraft's taxiing at a steady speed on a taxiway. This network was introduced in the paper "Verification of Image-based Neural Network Controllers Using Generative Models" [68]. The neural network integrates a concatenation of the cGAN (conditional GAN) and controller, resulting in a unified neural network controller with low-dimensional state inputs. In this problem, the inputs to the neural network consist of two state variables and two latent variables. The aircraft's state is determined by its crosstrack position (p) and heading angle error (θ) with respect to the taxiway center line. Two latent variables with a range of -0.8 to 0.8 are introduced to account for environmental changes. Because in this case the output spec depends on both the input and output and considering the VNN-LIB limitation, we added a skip-connection layer to the neural network to have the input values present in the output space. We also added one linear layer after that to create a linear equation for each local model.

Specifications As mentioned earlier, the aim of this benchmark is to examine whether the neural network output stays within the range defined by the linear model's output, including a margin for uncertainty.Given input $x \in X$ and output $Y = f_{NN}(x)$, the query is of the form: $A_{mat} \times X + b + U_{lb} \leq Y \leq A_{mat} \times X + b + U_{ub}$ for each linear model in its abstraction region.

Link https://github.com/aliabigdeli/LinearizeNN_benchmark2024

5.4 ml4acopf

Proposed by Haoruo Zhao, Michael Klamkin, Mathieu Tanneau, Wenbo Chen, and Pascal Van Hentenryck (Georgia Institute of Technology), and Hassan Hijazi, Juston Moore, and Haydn Jones (Los Alamos National Laboratory).

Motivation Machine learning models are utilized to predict solutions for an optimization model known as AC Optimal Power Flow (ACOPF) in the power system. Since the solutions are continuous, a regression model is employed. The objective is to evaluate the quality of these machine learning model predictions, specifically by determining whether they satisfy the constraints of the optimization model. Given the challenges in meeting some constraints, the goal is to verify whether the worst-case violations of these constraints are within an acceptable tolerance level.

Networks The neural network designed comprises two components. The first component predicts the solutions of the optimization model, while the second evaluates the violation of each constraint that needs checking. The first component consists solely of general matrix multiplication (GEMM) and rectified linear unit (ReLU) operators. However, the second component has a more complex structure, as it involves evaluating the violation of AC constraints using nonlinear functions, including sigmoid, quadratic, and trigonometric functions such as sine and cosine. This complex evaluation component is incorporated into the network due to a limitation of the VNNLIB format, which does not support trigonometric functions. Therefore, these constraints violation evaluation are included in the neural network.

Specifications In this benchmark, four different properties are checked, each corresponding to a type of constraint violation:

1. Power balance constraints: the net power at each bus node is equal to the sum of the power flows in the branches connected to that node.
2. Thermal limit constraints: power flow on a transmission line is within its maximum and minimum limits.
3. Generation bounds: a generator's active and reactive power output is within its maximum and minimum limits.
4. Voltage magnitude bounds: a voltage's magnitude output is within its maximum and minimum limits.

The input to the model is the active and reactive load. The chosen input point for perturbation is a load profile for which a corresponding feasible solution to the ACOPF problem is known to exist. For the feasibility check, the input load undergoes perturbation. Although this perturbation does not exactly match physical laws, the objective is to ascertain whether a machine learning-predicted solution with the perturbation can produce a solution that does not significantly violate the constraints.

The scale of the perturbation and the violation threshold are altered by testing whether an adversarial example can be easily found using projected gradient descent with the given perturbation. The benchmark, provided with a fixed random seed, is robust against the simple projected gradient descent that is implemented.

Link https://github.com/AI4OPT/ml4acopf_benchmark

5.5 ViT

Proposed by the α,β-CROWN team.

Motivation Transformers [125] based on the self-attention mechanism have much more complicated architectures and contain more kinds of nonlinerities, compared to simple feedforward networks with relatively simple activation functions. It makes verifying Transformers challenging. We aim to encourage the development of verification techniques for Transformer-based models, and we also aim to benchmark neural network verifiers on relatively complicated neural network architectures and more general nonlinearities. Therefore, we propose a new benchmark with Vision Transformers (ViTs) [36]. This benchmark is developed based on our work on neural network verification for models with general nonlinearities [105].

Networks The benchmark contains two ViTs, as shown in Table 5. Considering the difficulty of verifying ViTs, we modify the ViTs and make the models relatively shallow and narrow, with significantly reduced number of layers and attention heads. Following [107], we also replace the layer normalization with batch normalization. The models are mainly trained with PGD training [93], and we also add a weighted IBP [54, 106] loss for one of the models as a regularization.

Specifications The specifications are generated from the robustness verification problem with ℓ_∞ perturbation. We use the CIFAR-10 dataset with perturbation size $\epsilon = \frac{1}{255}$ at test time. We have filtered the CIFAR-10 test set to exclude instances where either adversarial examples can be found (by PGD attack [93] with 100 steps and 1000 restarts) or the vanilla CROWN-like method [138, 107] can already easily verify. We randomly keep 100 instances for each model, with a timeout threshold of 100 seconds. Note that since instances with adversarial examples

Table 5: Networks in the ViT benchmark.

Model	PGD_2_3_16	IBP_3_3_8
Layers	2	3
Attention heads	3	3
Patch size	16	8
Weight of IBP loss	0	0.01
Training ϵ	$\frac{2}{255}$	$\frac{1}{255}$
Clean accuracy	59.78%	62.21%

have mostly been excluded during the filtering process, this version of the benchmark may not be able to reflect soundness issues in verifiers, and we refer readers to [142] for discussions on testing soundness with models including ViT.

Link https://github.com/shizhouxing/ViT_vnncomp2023

5.6 LSNC-ReLU

Proposed by the α,β-CROWN team.

Motivation We develop a benchmark for the problem of verifying the Lyapunov stability of NN controllers in nonlinear dynamical systems within a region-of-intrest and a region-of-attraction. This is important for providing stability guarantees that are essential for safety-critical applications with NN controllers. It is also a useful application of neural network verification as recently demonstrated in [133, 104], and we refer readers to those works for more details on the problem.

Networks and Specifications The NN controller and Lyapunov function are trained using the technique presented in CT-BaB [104]. Compared to a general nonlinear dynamics as in [133], we utilized a neural-network dynamics. More specifically, the dynamics of this model is given by a shallow neural network with ReLU activations that are trained to approximate the 2D quadrotor dynamics. In this way, only ReLU nonlinearity is presented in the network (no trig functions, leakyrelu, etc.). The verification goal is to certify the Lyapunov stability of the NN controller in a nonlinear dynamical system within a certain Lyapunov sublevel set ($V < c$). Specifications for the benchmark are randomly generated and consist of random rings of the form $x : c_1 < V(x) < c_2$.

Link https://github.com/xiangruzh/LSNC-ReLU

5.7 Collins-RUL-CNN

Proposed by Collins Aerospace, Applied Research & Technology (website).

Motivation Machine Learning (ML) is a disruptive technology for the aviation industry. This particularly concerns safety-critical aircraft functions, where high-assurance design and verification methods have to be used in order to obtain approval from certification authorities for the new ML-based products. Assessment of correctness and robustness of trained models, such as neural networks, is a crucial step for demonstrating the absence of unintended functionalities [50, 76]. The key motivation for providing this benchmark is to strengthen the interaction between the VNN community and the aerospace industry by providing a realistic use case for neural networks in future avionics systems [75].

Application Remaining Useful Life (RUL) is a widely used metric in Prognostics and Health Management (PHM) that manifests the remaining lifetime of a component (e.g., mechanical bearing, hydraulic pump, aircraft engine). RUL is used for Condition-Based Maintenance (CBM) to support aircraft maintenance and flight preparation. It contributes to such tasks as augmented manual inspection of components and scheduling of maintenance cycles for components, such as repair or replacement, thus moving from preventive maintenance to *predictive* maintenance (do maintenance only when needed, based on component's current condition and estimated future condition). This could allow to eliminate or extend service operations and inspection periods, optimize component servicing (e.g., lubricant replacement), generate inspection and maintenance schedules, and obtain significant cost savings. Finally, RUL function can also be used in airborne (in-flight) applications to dynamically inform pilots on the health state of aircraft components during flight. Multivariate time series data is often used as RUL function input, for example, measurements from a set of sensors monitoring the component state, taken at several subsequent time steps (within a time window). Additional inputs may include information about the current flight phase, mission, and environment. Such highly multi-dimensional input space motivates the use of Deep Learning (DL) solutions with their capabilities of performing automatic feature extraction from raw data.

Networks The benchmark includes 3 convolutional neural networks (CNNs) of different complexity: different numbers of filters and different sizes of the input space. All networks contain only convolutional and fully connected layers with ReLU activations. All CNNs perform the regression function. They have been trained on the same dataset (time series data for mechanical component degradation during flight).

Specifications We propose 3 properties for the NN-based RUL estimation function. First, two properties (robustness and monotonicity) are local, i.e., defined around a given point. We provide a script with an adjustable random seed that can generate these properties around input points randomly picked from a test dataset. For robustness properties, the input perturbation (delta) is varied between 5% and 40%, while the number of perturbed inputs varies between 2 and 16. For monotonicity properties, monotonic shifts between 5% and 20% from a given point are considered. Properties of the last type ("if-then") require the output (RUL) to be in an expected value range given certain input ranges. Several if-then properties of different complexity are provided (depending on range widths).

Link https://github.com/loonwerks/vnncomp2022

Paper Available in [75] or on request.

5.8 VGGNET16

Proposed by Stanley Bak, Stony Brook University

Motivation This benchmark tries to scale up the size of networks being analyzed by using the well-studied VGGNET-16 architecture [110] that runs on ImageNet. Input-output properties are proposed on pixel-level perturbations that can lead to image misclassification.

Networks All properties are run on the same network, which includes 138 million parameters. The network features convolution layers, ReLU activation functions, as well as max pooling layers.

Specifications Properties analyzed ranged from single-pixel perturbations to perturbations on all 150528 pixles (L-infinity perturbations). A subset of the images was used to create the specifications, one from each category, which was randomly chosen to attack. Pixels to perturb were also randomly selected according to a random seed.

Link https://github.com/stanleybak/vggnet16_benchmark2022/

5.9 Traffic Signs Recognition

Proposed by Mădălina Eraşcu and Andreea Postovan (West University of Timisoara, Romania)

Motivation Traffic signs play a crucial role in ensuring road safety and managing traffic flow in both city and highway driving. The recognition of these signs, a vital component of autonomous driving vision systems, faces challenges such as susceptibility to adversarial examples [113] and occlusions [139], stemming from diverse traffic scene conditions.

Networks Binary neural networks (BNNs) show promise in computationally limited and energy-constrained environments within the realm of autonomous driving [62]. BNNs, where weights and/or activations are binarized to ± 1, offer reduced model size and simplified convolution operations for image recognition compared to traditional neural networks (NNs).

We trained and tested various BNN architectures using the German Traffic Sign Recognition Benchmark (GTSRB) dataset [3]. This multi-class dataset, containing images of German road signs across 43 classes, poses challenges for both humans and models due to factors like perspective change, shade, color degradation, and lighting conditions. The dataset was also tested using the Belgian Traffic Signs [1] and Chinese Traffic Signs [2] datasets. The Belgium Traffic Signs dataset, with 62 classes, had 23 overlapping classes with GTSRB. The Chinese Traffic Signs dataset, with 58 classes, shared 15 classes with GTSRB. Pre-processing steps involved relabeling classes in the Belgium and Chinese datasets to match those in GTSRB and eliminating non-overlapping classes (see [98] for details).

We provide three models with the structure in Figures 1, 2, and 3. They contain QConv, Batch Normalization (BN), Max Pooling (ML), Fully Connected/Dense (D) layers. Note that the QConv layer binarizes the corresponding convolutional layer. All models were trained for 30 epochs. The model from Figure 1 was trained with images having the dimension 64px x 64 px, the one from Figure 2 with 48px x 48 px and the one from Figure 3 with 30px x 30 px. The two models involving Batch Normalization layers introduce real valued parameters besides the binary ones, while the third one contains only binary parameters (see Table 6) for statistics.

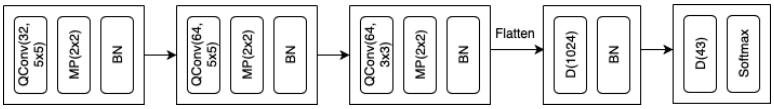


Figure 1: Accuracy Efficient Architecture for GTSRB and Belgium dataset

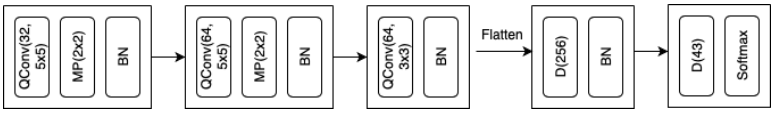


Figure 2: Accuracy Efficient Architecture for Chinese dataset

Specifications To evaluate the *adversarial robustness* of the networks above, we assessed perturbations within the infinity norm around zero, with the radius denoted as $\epsilon = \{1, 3, 5, 10, 15\}$. This involved randomly selecting three distinct images from the GTSRB dataset's test set for each model and generating VNNLIB files for each epsilon in the set. In total, we created

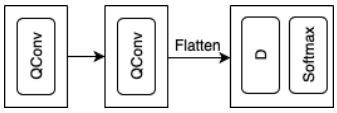


Figure 3: XNOR(QConv) architecture

Table 6: Training and Testing Statistics

Input size	Model name	Accuracy			#Params		
		German	China	Belgium	Binary	Real	Total
64px × 64px	Figure 1	96.45	81.50	88.17	1772896	2368	1775264
48px × 48px	Figure 2	95.28	83.90	87.78	904288	832	905120
30px × 30px	Figure 3	81.54	N/A	N/A	1005584	0	1005584

45 VNNLIB files. Due to a 6-hour total timeout constraint for solving all instances, each instance had a maximum timeout of 480 seconds. To review the generated VNNLIB specification files submitted to VNNCOMP 2023, as well as to generate new ones, please refer to https://github.com/apostovan21/vnncomp2023.

Link https://github.com/apostovan21/vnncomp2023

5.10 CIFAR100

Proposed by the α,β-CROWN team.

Motivation This benchmark is reused from VNN-COMP 2022 with a reduced complexity (only two out of the four models with medium sizes are retained). See details in Section 4.5 of the report of VNN-COMP 2022 [97].

Networks We provide two ResNet models on CIFAR-100 with different model widths and depths (input dimension $32 \times 32 \times 3$, 100 classes):

- `CIFAR100-ResNet-medium`: 8 residual blocks, 17 convolutional layers + 2 linear layers
- `CIFAR100-ResNet-large`: 8 residual blocks, 19 convolutional layers + 2 linear layers (almost identical to standard ResNet-18 architecture)

Specifications We randomly select 100 images from the CIFAR-100 test set with a verification timeout of 100 seconds for each of the two models. We filtered out the samples which can be verified by vanilla CROWN (which is used during training) to make the benchmark more challenging. The filtering process is done offline on a machine with a GPU due to the large sizes of these models. A small proportion of instances (around 18%) with adversarial examples have been retained for potentially identifying unsound results.

Link https://github.com/huanzhang12/vnncomp2024_cifar100_benchmark

5.11 TinyImagenet

Proposed by the α,β-CROWN team.

Motivation This benchmark is reused from VNN-COMP 2022. See details in Section 4.5 of the report of VNN-COMP 2022 [97].

Networks We provide a ResNet for TinyImageNet (input dimension $64 \times 64 \times 3$, 200 classes):

- `TinyImageNet-ResNet-medium`: 8 residual blocks, 17 convolutional layers + 2 linear layers

Specifications We randomly select 200 images from the TinyImageNet test set with a verification timeout of 100 seconds for each of the two models. A filtering procedure has been adopted similar to the CIFAR100 benchmark.

Link https://github.com/huanzhang12/vnncomp2024_tinyimagenet_benchmark

5.12 Yolo

Proposed by the α,β-CROWN team.

Motivation Object detection is central to many safety-critical applications, yet verification research has had limited benchmarks that capture its unique challenges. Even in a simplified form, YOLO highlights key aspects of detection tasks: multiple outputs, spatial reasoning over bounding boxes, and the need to ensure reliable perception under perturbations. A YOLO benchmark therefore provides a concrete step toward advancing verification methods for complex perception tasks such as object detection.

Networks The model is a modified version of YOLOv2. For simplicity of verification, the backbone of the original YOLOv2 model is replaced with a much smaller network. The network takes in an image ($3 \times 52 \times 52$) and outputs a tensor of a shape $125 \times 13 \times 13$, which contains the confidence scores, the classification result, and the positions of 5 bounding boxes for each of the 13×13 grid cells. The network was trained using adversarial training, incorporating an ℓ_1 regularization term in the loss.

Specifications Robustness properties are defined on the inputs and raw output tensors of the backbone network. For each instance, the property is verified if there is at least one bounding box successfully predicting the same object (with a confidence score greater than a preset threshold) given any bounded random perturbations on the input image.

Link https://github.com/xiangruzh/Yolo-Benchmark

5.13 SoundnessBench

Proposed by the α,β-CROWN team.

Motivation We introduce SoundnessBench, a novel benchmark designed to evaluate the soundness of neural network (NN) verifiers, inspired by the work [142]. While recent years have seen significant progress in developing both NN verifiers and benchmarks, existing benchmarks typically lack ground truth for challenging instances. As a result, it remains difficult to determine whether a verification claim is sound when no verifier can falsify the property under consideration.

Networks SoundnessBench includes a simple feedforward neural network and a residual network with minimal architectures and standard operators widely supported by mainstream verifiers. These operators include fully connected (linear) layers, 2D convolutional layers, ReLU activations, as well as reshaping and flattening operations between 2D and 1D formats.

Specifications The verification tasks are synthetic, with some specifications intentionally designed to embed counterexamples that are difficult for standard falsification methods—such as projected gradient descent (PGD)—to discover. Specifications are expressed as conjunctions of clauses.

Link https://github.com/Kevinyang555/SoundnessBench_vnncomp2026

5.14 Relusplitter

Proposed by Linhan Li and ThanhVu Nguyen (George Mason University)

Motivation Many existing DNN verification benchmarks are becoming ineffective at evaluating modern verifiers because their instances are often too easy or too hard and lack architectural diversity. To address these issues and allow for the reuse of established benchmarks, we propose Relusplitter [88]. It takes a DNN verification instance (network and property) and generates a semantically equivalent network on which the property is harder to verify. Relusplitter achieves this by systematically destabilizing stable neurons in the network.

Description This benchmark uses selected instances from benchmarks used in previous VNNCOMP iterations as seed inputs to generate more challenging instances. The networks include both fully connected (FC) and convolutional (Conv) architectures. All networks use a standard feed-forward structure. The timeout for the instances was reduced based on results from prior iterations, and the generated instances have 3x timeout. Benchmarks used as seed inputs include (1) ACAS Xu, (2) MNIST_FC, (3) Oval21, and (4) Cifar Biasfield.

Networks The seed instances use the original network, and the generated instances use a destabilized network generated from the original network.

Specifications The original specification from the seed instance was used for each generated instance.

Link https://github.com/dynaroars/relusplitter/tree/VNNCOMP-25

5.15 TLL Verify Bench

Proposed by James Ferlez (University of California, Irvine)

Motivation This benchmark consists of Two-Level Lattice (TLL) NNs, which have been shown to be amenable to fast verification algorithms (e.g. [52]). Thus, this benchmark was proposed as a means of comparing TLL-specific verification algorithms with general-purpose NN verification algorithms (i.e. algorithms that can verify arbitrary deep, fully-connected ReLU NNs).

Networks The networks in this benchmark are a subset of the ones used in [52, Experiment 3]. Each of these TLL NNs has $n = 2$ inputs and $m = 1$ output. The architecture of a TLL NN is further specified by two parameters: N, the number of local linear functions, and M, the number of selector sets. This benchmark contains TLLs of sizes $N = M = 8, 16, 24, 32, 40, 48, 56, 64$, with 30 randomly generated examples of each (the generation procedure is described in [52, Section 6.1.1]). At runtime, the specified verification timeout determines how many of these networks are included in the benchmark so as to achieve an overall 6-hour run time; this selection process is deterministic. Finally, a TLL NN has a natural representation using multiple computation paths [52, Figure 1], but many tools are only compatible with fully-connected networks. Hence, the ONNX models in this benchmark implement TLL NNs by "stacking" these computation paths to make a fully connected NN (leading to sparse weight matrices: i.e. with many zero weights and biases). The `TLLnet` class (https://github.com/jferlez/TLLnet) contains the code necessary to generate these implementations via the `exportONNX` method.

Specifications All specifications have as input constraints the hypercube $[-2, 2]^2$. Since all networks have only a single output, the output properties consist of a randomly generated real number and a randomly generated inequality direction. Random output samples from the network are used to roughly ensure that the real number property has an equal likelihood of being within the output range of the NN and being outside of it (either above or below all NN outputs on the input constraint set). The inequality direction is generated independently and

with each direction having an equal probability. This scheme biases the benchmark towards verification problems for which counterexamples exist.

Link https://github.com/jferlez/TLLVerifyBench

Commit 199d2c26d0ec456e62906366b694a875a21ff7ef

5.16 ACAS Xu

Networks The ACASXu benchmark consists of ten properties defined over 45 neural networks used to issue turn advisories to aircraft to avoid collisions. The neural networks have 300 neurons arranged in 6 layers, with ReLU activation functions. There are five inputs corresponding to the aircraft states, and five network outputs, where the minimum output is used as the turn advisory the system ultimately produces.

Specifications We use the original 10 properties [66], where properties 1-4 are checked on all 45 networks as was done in later work by the original authors [67]. Properties 5-10 are checked on a single network. The total number of benchmarks is therefore 186. The original verification times ranged from seconds to days—including some benchmark instances that did not finish. This year we used a timeout of around two minutes (116 seconds) for each property, in order to fit within a total maximum runtime of six hours.

5.17 Real-world distribution shifts

Proposed by the Marabou team.

Motivation While robustness against handcrafted perturbations (e.g., norm-bounded) for perception networks are more commonly investigated, robustness against real-world distribution shifts [130] are less studied but of practical interests. This benchmark set contains queries for verifying the latter type of robustness.

Networks The network is a concatenation of a generative model and a MNIST classifier. The generative model is trained to take in an unperturbed image and an embedding of a particular type of distribution shifts in latent space, and produce a perturbed image. The distribution shift captured in this case is the "shear" perturbation.

Specifications The verification task is to certify that a classifier correctly classifies all images in a perturbation set, which is a set of images generated by the generative model given a fixed image and a ball centering the mean perturbations on this image (in the latent space). This mean perturbation is computed by a prior network.

Link https://github.com/wu-haoze/dist-shift-vnn-comp

5.18 safeNLP

Proposed by Marco Casadio, Ekaterina Komendantskaya, Luca Arnaboldi, Tanvi Dinkar.

Motivation While considerable research has been dedicated to the verification of DNN-based systems in domains such as computer vision, there has been a notable lack of focus on the verification of natural language processing (NLP) systems. This is particularly critical given the rise of conversational agents across various domains, where inaccurate or misleading responses can cause real-world harm. For example, recent EU legislation [82] requires chatbots to disclose their non-human nature when queried, and developers of the chatbots should provide firm, and if possible, formal, guarantees that such disclosure will be given in an accurate manner. Medical assistants give another example where formal guarantees about the conversational

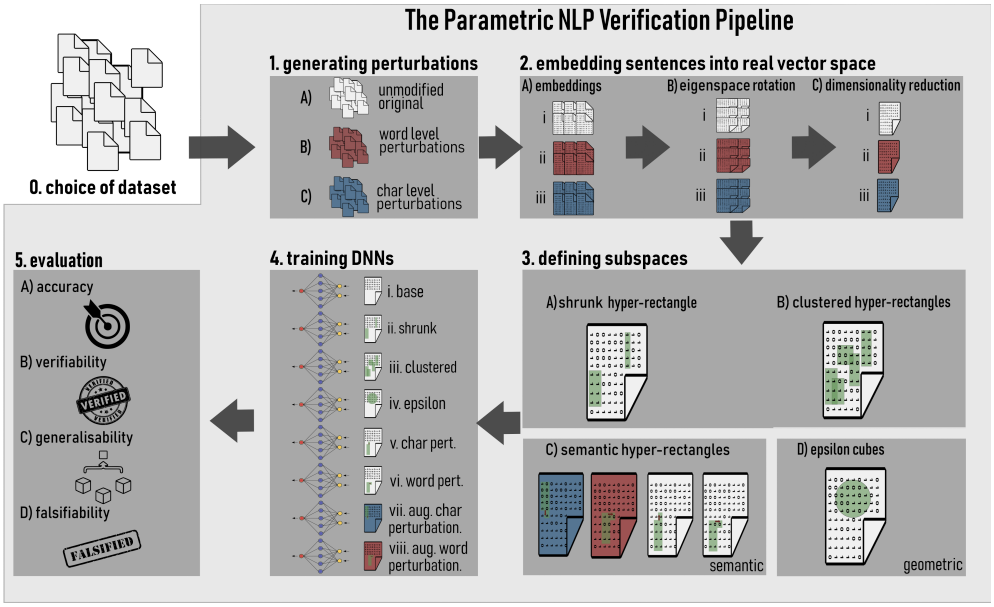


Figure 4: *Generic approach to generating the NLP verification pipelines [27, 28] deployed to obtain the safeNLP benchmark.*

agent responses are needed in order to safeguard against chatbots generating harmful medical advice [20]. While some initial work has been done in this area of NLP verification [64, 61, 128, 140, 127, 77, 37, 108, 21], no agreement on commonly accepted benchmarks has been reached in this domain. To address this gap, we introduce safeNLP, the first such benchmark.

Application In [28], we have undertaken a large-scale study of the existing literature on NLP verification, and distilled common patterns among the existing approaches. Usually, given a dataset consisting of sentences divided into classes, Large Language Models (LLMs) are used to embed these sentences into real-vector spaces, after which smaller neural networks are trained to classify the embedded vectors (relative to the originally given classes). For verification, one can generate meaning-preserving sentence perturbations, again embed them into vector spaces, and verify that subspaces that contain the (embeddings of) the perturbed sentences are classified correctly. Also, in line with classical verification pipelines [29], one can use these input subspaces to train the neural network to be robust on them. The problem was that each of the existing approaches [64, 61, 128, 140, 127, 77, 37, 108, 21] used parts of this pipeline in different ways, which made it difficult to compare or audit the results. In [27, 28], we made a generic implementation of this pipeline, where each of the components of the pipeline is implemented in a modular and transparent way. For example, we can choose and vary embedding functions, training modes, algorithms for sentence perturbations and algorithms for robust training, independently and modularly; as shown in Figure 4. This implementation was used to generate the presented VNNCOMP benchmark.

- *Datasets:* Although there was no clear consensus in [64, 61, 128, 140, 127, 77, 37, 108, 21], the most frequently used dataset in prior works was the IMDB dataset containing film reviews. Its disadvantage is unclear relation to safety critical domains that usually motivate verification efforts. On the other hand, none of the previously used datasets concerned safety-critical applications of NLP. We decided to address this problem, and

therefore applied our generic NLP verification pipeline on two safety-critical datasets: R-U-A-Robot [55], which focuses on the chatbot disclosure problem, and Medical [9], which addresses the issue of harmful advice provided by medical chatbots. Both datasets are pre-processed into two classes, positive and negative, to simplify the verification task. For further details on the pre-processing steps and datasets, see [28] and the benchmark GitHub repository.

- *Input Space:* In both datasets, sentences are transformed into fixed-size vector representations, i.e. embeddings, which serve as the inputs to the neural networks. For this VNNCOMP benchmark, we used Sentence-BERT [99].

- *What to Verify:* For each dataset, we generated meaning-preseving sentence perturbations at character and word level as in Moradi et al. [96] and at sentence level with Vicuna [31]. For each positive sentence in the dataset, the smallest hypercube containing the embeddings of all of its obtained perturbations formed one input subspace for verification. Such subspaces were obtained for all positive sentences from the given data set, and were subject to VNCCOMP verification challenge.

- *A note on broader impact:* Verified models can serve as filters for larger NLP systems: e.g. to screen inputs to ensure they meet safety criteria before being passed on to more complex models.

Networks The safeNLP benchmark includes two neural networks, each corresponding to a different dataset (R-U-A-Robot and Medical). Both networks share the same architecture, consisting of two fully-connected layers. The hidden layer has 128 units with a ReLU activation function, while the output layer has 2 units representing the two classification classes (positive/negative). To enhance the robustness of the networks to the specified safety requirements, they are trained using a custom PGD (Projected Gradient Descent) [94] adversarial training technique. In particular, the PGD attack explores the above-mentioned subspaces of the input space (cf. also Figure 4).

Specifications The benchmark uses hyper-rectangles in the 30-dimensional embedding space as the subspaces of choice, offering a computationally efficient way to define more precise and adaptable regions compared to the traditional ϵ-cubes. The specifications require verifying that, for a given network and hyper-rectangle, every point within the hyper-rectangle is classified as the positive class by the network. To meet time constraints, we randomly select 1,080 such specifications, each linked to one of the two networks and a corresponding hyper-rectangle, with a timeout of 20 seconds per specification.

Link `https://github.com/ANTONIONLP/safeNLP`

5.19 CORA Benchmark

Proposed by the CORA team.

Motivation The verification of neural networks can be quite slow, i.e., the verification of a single instance can take multiple days – which is often hard to justify, particularly in safety-critical scenarios. To encourage the fast verification of neural networks, our benchmark focuses on the verification time by setting a small timeout and testing three different (adversarial) training techniques that aim to ease the verifiability.

Networks The benchmark consists of one ReLU-neural network architecture (7x250 + ReLU), which was trained on three datasets, (MNIST, SVHN, and CIFAR10), using three different (adversarial) training methods, i.e., standard (point), interval-bound propagation, and set-based. Both interval-bound propagation and set-based training are training methods that improve the robustness of the trained neural network and aim to ease later verification. The neural networks are taken from the first evaluation run of [79]; please refer to [79] for the training details.

Specifications All networks are trained on classification tasks. The goal is to verify that no image within a given input set is incorrectly classified.

Link https://github.com/kollerlukas/cora-vnncomp2024-benchmark

5.20 SAT ReLU

Proposed by Edoardo Manino (The University of Manchester)

Motivation Neural network verifiers are fundamentally not different from any other piece of software in that they may contain bugs. As such, it is necessary to test their behavior for correctness. To this end, the SAT ReLU benchmark is constructed with the additional requirement of having its verification verdicts (safe or unsafe) known a priori. With it, we can test all neural network verifiers for correctness.

Description The SAT ReLU benchmark is entirely synthetic. Each instance is built by generating a random 3-SAT boolean formula with known satisfiability verdict. In total, we generate 50 satisfiable formulae and 50 unsatisfiable ones in CNF format. We vary the number of variables in $v = [2, 100]$ and the number of clauses in $c = [v, 5v]$. Then, we encode the formulae as neural network verification problems.

Networks We encode each CNF formula as a fully-connected ReLU network with one hidden layer. To do so, we take inspiration from the ReLU gadgets introduced by [66]. First, we assume that each element of the input $x \in \mathbb{R}^v$ represents a boolean variable $x_i \in \{0, 1\}$. Half of the network encodes the boolean arithmetic necessary to compute the CNF formula under this assumption. Second, we let each variable vary in $x_i \in [0, 1]$ and use the other half of the network to encode the necessary binarization constraints. Finally, we merge the two halves into a single network. The final network has v inputs, $c + 2v$ hidden activations, and 2 outputs.

Specifications Given the nature of the networks we construct, the original CNF formula is satisfiable if and only if there exists some input $x \in \{0, 1\}^n$ that produces an output of $y_1 \geq 1$. Non-binary inputs cause the second output to be positive, and thus are rejected by setting a binarization constraint of $y_2 \leq 0$.

Link https://github.com/emanino/neurocodebench

5.21 MetaRoom

Proposed by Hanjiang Hu (CMU), Zuxin Liu (CMU), Linyi Li (UIUC), Jiacheng Zhu (CMU), Ding Zhao (CMU).

Motivation MetaRoom Benchmark is based on the MetaRoom dataset, which aims at the robustness certification and verification of deep learning based vision models against camera motion perturbation in the indoor robotics application. Specifically, this benchmark focuses on the classification of 20 labels against camera moves within tiny perturbation radii of translation along the z-axis and rotation along the y-axis (e.g. 1e-5 m, 2.5e-4 degree). The MetaRoom dataset is a realistic indoor object recognition dataset with dense point cloud maps, which

enables image projections from different camera poses and simulates object-centric semantic robustness in real-world robotic applications. More details about the dataset can be found in [59].

Networks The models of `cnn_4layer` and `cnn_6layer` are 4-layer and 6-layer convolutional neural networks with a last layer of a fully connected layer for multi-class classification. The input shape is `1*3*32*56`, which is projected from one-dimensional relative camera rotation or translation around 0 (e.g. [-1e-5, 1e-5]) from a very dense point cloud based on direct Z-buffering. And the output is for the classification of 20 labels. The pretrained models give correct predictions at camera motion movement origins.

Specifications Simplified from the 1-dim semantic perturbation studied in [58, 60], the benchmark for VNN-COMP adopts the pre-projected image specifications from a very dense point cloud, where the lower and upper bounds are all pixels of the image during the projection within the camera perturbation radius of translation along the z-axis or rotation along the y-axis. The output specifications will be the classification labels at camera movement origins.

Link https://github.com/HanjiangHu/metaroom_vnn_comp2023

5.22 cersyve

Proposed by Yujie Yang (THU, CMU), Hanjiang Hu (CMU), Tianhao Wei (CMU), Shengbo Eben Li (THU), Changliu Liu (CMU).

Motivation Neural safety certificate verification is a special type of verification problems that require neural networks to satisfy certain properties everywhere in the state space, i.e., the input set of the verification problem is the entire state space. This is different from most existing benchmarks, which only consider verification in either a small disturbance set around data samples or part of the state space, such as MNIST, CIFAR, and ACAS Xu. Another distinct feature of safety certificate verification is that it not only involves a single safety certificate network but also needs system dynamics and control policies for verification, and certain conversions are required before these components can be formulated as a standard verification problem.

Description While this version of cersyve only contains necessary networks and specifications for verification, our full version also includes a set of neural safety certificate synthesis tools, including pre-training, adversarial training, and verification-guided training modules, as well as evaluation tools for synthesized certificates. For a detailed description of cersyve, please refer to our paper [134].

Networks The cersyve benchmark contains nine commonly used control tasks with state dimensions ranging from two to six. These tasks include both linear and nonlinear dynamics and safety constraints. We provide two ONNX models for each task. One is a pre-trained safety certificate, and the other is fine-tuned. The models are already integrated with all necessary elements for verification, such as system dynamics, constraints, and control policies. Users can view each of them as a single neural network for standard verification.

Specifications The cersyve benchmark requires to verify that within a bounded state space, whether the constraint satisfaction and forward invariance properties of a neural safety certificate is satisfied for all states. For a detailed description of the verification problem formulation, please refer to our paper [134]. The input specifications are lower and upper bounds on each dimension of the state. The output specifications are always two inequalities corresponding to the two properties.

Link https://github.com/intelligent-control-lab/cersyve.jl/tree/VNN-COMP-2025

Table 7: Performance and complete-verification difficulty of the models underlying the Challenging Certified Training benchmark. Verification results were obtained with α, β-CROWN using a 1,000-second cutoff on 10,000 test-set properties per model.

Dataset / ϵ	Network	Standard acc.	Certified acc.	Timeouts
CIFAR-10, 2/255	CNN-7	83.54%	66.04%	658 (6.58%)
CIFAR-10, 2/255	Wide CNN-7	85.37%	60.28%	1,433 (14.33%)
CIFAR-10, 8/255	CNN-7	57.19%	35.49%	134 (1.34%)
CIFAR-10, 8/255	Wide CNN-7	57.74%	34.60%	132 (1.32%)
TinyImageNet, 1/255	CNN-7	41.21%	27.81%	236 (2.36%)
TinyImageNet, 1/255	Wide CNN-7	41.89%	28.35%	401 (4.01%)
			Total timeouts	2,994 (4.99%)

5.23 Challenging Certified Training

Proposed by Konstantin Kaulen (RWTH Aachen University)

Motivation Using recent advances in certified training, we obtained state-of-the-art certifiably robust image classifiers using the CTRAIN library for certified training [69, 73]. Their standard and certified accuracies are summarized in Table 7. Despite this progress, we observed that complete verification of these models is highly computationally demanding. When evaluating all test-set properties using α, β-CROWN with a cutoff of 1,000 seconds, 2,994 out of 60,000 verification tasks timed out, with a timeout rate of up to 14.33%. Thus, complete verification is becoming a major computational bottleneck for further progress in certifiably robust neural networks: improvements in certified training performance are of limited practical value if the resulting guarantees cannot be established within reasonable computational budgets. The Challenging Certified Training benchmark therefore provides difficult and practically relevant instances for measuring progress in complete neural network verification.

Description The benchmark contains 60 local-robustness instances derived from the six models shown in Table 7. To obtain instances spanning a broad range of verification difficulty, all 10,000 test-set properties for each model were first evaluated using α, β-CROWN with its standard configuration and a cutoff time of 1,000 seconds. The resulting instances were stratified according to their observed verification runtime into four categories: $[0, 10]$ seconds, $[10, 100]$ seconds, $[100, 1000]$ seconds, and timeouts. For each model, we sample two instances from each of the first two categories and three instances from each of the latter two categories, yielding ten properties per model and 60 benchmark instances overall. This construction combines moderately difficult properties with instances that remain unsolved by a state-of-the-art verifier within a substantial computational budget.

Networks The benchmark comprises six convolutional neural networks trained using MTL-IBP [34]. For CIFAR-10, it contains CNN-7 architectures for ℓ_∞ perturbation radii of $\epsilon = 2/255$ and $\epsilon = 8/255$. For TinyImageNet, it contains corresponding standard and width-scaled CNN-7 architectures for $\epsilon = 1/255$. Their standard and certified accuracies are reported in Table 7.

Specifications Each specification encodes standard ℓ_∞ local robustness of a CIFAR-10 or TinyImageNet test image at the perturbation radius corresponding to the respective model.

Link https://github.com/kkaulen/vnncomp_challenging_certified_training

5.24 Additional Benchmarks

We have not yet obtained benchmark descriptions for other benchmark sets reported in Table 4. We will update the report when these descriptions are available. Artifacts of the benchmarks are available in the repository[6].

[6] https://github.com/VNN-COMP/vnncomp2026_benchmarks

6 Results

Each tool was run on each of the benchmarks and produced a `csv` result file, that was provided as feedback to the tool authors using the online execution platform. The final `csv` files for each tool as well as scoring scripts are available online: https://github.com/VNN-COMP/vnncomp2026_results. The results were automatically analyzed to compute scores and create the statistics presented in this section, where a tool is omitted from individual benchmark results if it timed out on all instances.

6.1 Regular Track

The regular track contained all the benchmarks that were voted for by at least half of all participants.

Table 8: Overall Score

#	Tool	Score
1	α-β-CROWN	2344.0
2	Vibecheck	2278.6
3	PyRAT	1813.6
4	NeuralSAT	1725.5
5	NNV	1149.5
6	CORA	1055.5
7	Bright	1000.5
8	N2v	984.5
9	SuperSimple	665.3
10	Act	650.3
11	nnenum	489.3
12	SobolBox	450.8
13	Agate	0

6.2 Extended Track

All benchmarks that were voted for by at least one team and did not make it into the regular track were part of the extended track. Every benchmark was voted for at least once, so no benchmark was unscored.

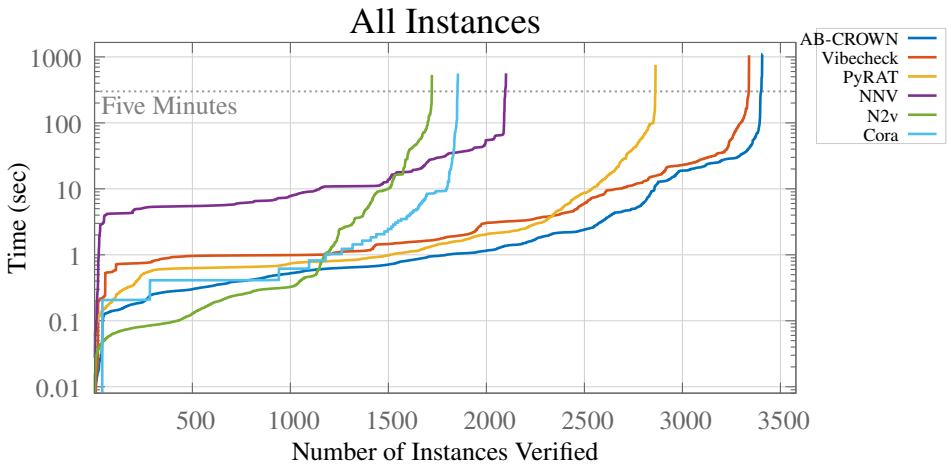


Figure 5: Cactus Plot for All Instances.

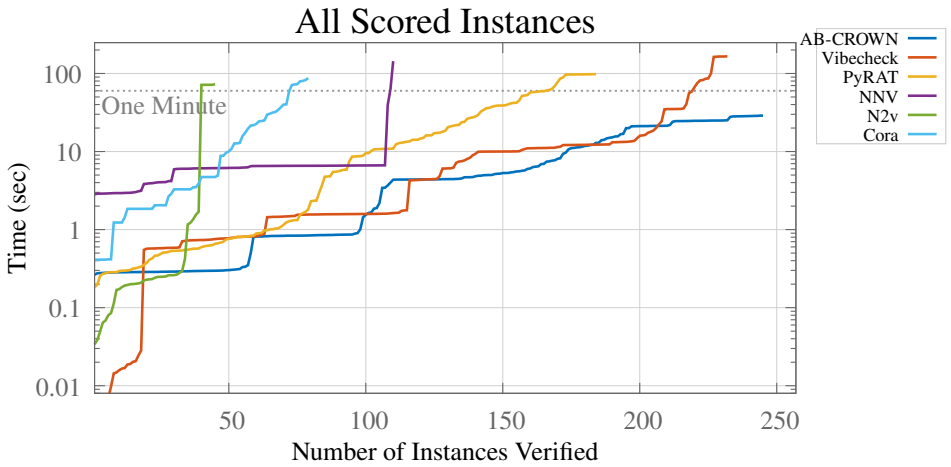


Figure 6: Cactus Plot for All Scored Instances.

Table 9: Overall Score

#	Tool	Score
1	α-β-CROWN	600.0
2	Vibecheck	574.0
3	PyRAT	478.0
4	NNV	271.3
5	N2v	178.0
6	CORA	158.0

6.3 Other Stats

This section presents other statistics related to the measurements that are interesting but did not play a direct role in scoring this year. The results reflect the tool performance on the regular track.

Table 10: Overhead

#	**Tool**	**Seconds**
1	nnenum	0.9
2	CORA	1.0
3	Vibecheck	1.7
4	Agate	2.7
5	Bright	2.9
6	NeuralSAT	3.2
7	SuperSimple	3.6
8	N2v	4.1
9	α-β-CROWN	5.6
10	SobolBox	7.4
11	PyRAT	7.5
12	NNV	8.2
13	Act	10.9

Table 11: Num Benchmarks Participated

#	**Tool**	**Count**
1	Vibecheck	25
2	PyRAT	24
3	α-β-CROWN	24
4	NeuralSAT	22
5	N2v	21
6	NNV	19
7	Agate	18
8	SuperSimple	17
9	nnenum	13
10	CORA	13
11	Act	12
12	SobolBox	11
13	Bright	11

Table 12: Num Instances Verified

#	Tool	Count
1	α-β-CROWN	3164
2	Vibecheck	3108
3	PyRAT	2679
4	NeuralSAT	2637
5	Bright	2048
6	NNV	1991
7	CORA	1776
8	N2v	1677
9	Act	1558
10	SuperSimple	1551
11	nnenum	1378
12	SobolBox	559
13	Agate	254

Table 13: Num SAT

#	Tool	Count
1	α-β-CROWN	1182
2	Vibecheck	1161
3	PyRAT	1093
4	N2v	1045
5	Bright	1022
6	SuperSimple	1007
7	NeuralSAT	998
8	NNV	963
9	CORA	898
10	Act	759
11	nnenum	695
12	SobolBox	355

Table 14: Num UNSAT

#	**Tool**	**Count**
1	α-β-CROWN	1982
2	Vibecheck	1947
3	NeuralSAT	1639
4	PyRAT	1586
5	NNV	1028
6	Bright	1026
7	CORA	878
8	Act	799
9	nnenum	683
10	N2v	632
11	SuperSimple	544
12	Agate	254
13	SobolBox	204

Table 15: Incorrect Results (or Missing CE)

#	**Tool**	**Count**
1	Agate	955
2	NeuralSAT	94
3	nnenum	58
4	SuperSimple	14
5	Act	10
6	SobolBox	2
7	CORA	1
8	α-β-CROWN	1

7 Conclusion and Ideas for Future Competitions

This report summarizes the 7$^{\text{th}}$ Verification of Neural Networks Competition (VNN-COMP), held in 2026. As per last year, the best-performing tools are GPU-enabled linear bound propagation methods using a branch-and-bound framework. The new VNNLIB 2.0 input format was widely adopted by tool proposers, which sends a positive signal. It shows that the community is eager to extend the reach of verification to more complex properties; we hope this will encourage difficult benchmarks propositions for the next iteration.

VNN-COMP 2026 successfully implemented a range of improvement opportunities identified during the previous iteration. These included requiring witnesses of found counter-examples to disambiguate tool disagreement, increasing automatization to enable a smoother final evaluation, and making a broader range of AWS instances available to allow for a better fit with tools' requirements. Additionally, we introduced the Competition Contribution papers included in the SAIV proceedings [6] for participants to summarize their approaches. In addition, in cases where the answer to a benchmark is known a priori, we will explore using the known correct results as the ground truth. Further ideas for future competitions include the use of scored benchmarks specifically designed for year-on-year progress tracking, the reduction of tool tuning, a batch-processing mode, and more rigorous soundness evaluation. Finally, as research on proof generation for neural network verification grows [63, 51, 44], a longer term goal is to require solvers to produce UNSAT proofs.

Acknowledgements

The 2026 competition was supported by SafeIntelligence and MathWorks, which provided funds for computation costs.

Additionally, this research was supported in part by the Air Force Research Laboratory Information Directorate, through the Air Force Office of Scientific Research Summer Faculty Fellowship Program, Contract Numbers FA8750-15-3-6003, FA9550-15-0001 and FA9550-20-F-0005. This material is based upon work supported by the Air Force Office of Scientific Research under award numbers FA9550-19-1-0288, FA9550-21-1-0121, and FA9550-22-1-0019, the National Science Foundation (NSF) under grant numbers 2107035, 2220401, 2220418, 2220426, and 2238133, and the Defense Advanced Research Projects Agency (DARPA) Assured Neuro Symbolic Learning and Reasoning (ANSR) programs through contract number and FA8750-23-C-0518. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force, DARPA, nor NSF. This research was additionally supported by a grant from the project AL 1185/33-1 funded by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG).

Tool and benchmark authors listed in Section 4 and Section 5 participated in the preparation and review of this report. Special thanks go to Volkan Şahin for his significant contributions to the development and maintenance of the evaluation system.

References

[1] Belgian Traffic Sign Database. https://www.kaggle.com/datasets/shazaelmorsh/trafficsigns. Accessed: March 25th, 2023.

[2] Chinese Traffic Sign Database. https://www.kaggle.com/datasets/dmitryyemelyanov/chinese-traffic-signs. Accessed: March 25th, 2023.

[3] German Traffic Sign Recognition Benchmark. https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign?datasetId=82373&language=Python. Accessed: March 25th, 2023.

[4] IACS Neural Network Verification System. https://github.com/iacs-csu-2020/INNVerS.

[5] SobolBox. https://github.com/dassarthak18/SobolBox. VNNCOMP-2025 branch.

[6] *AI Verification: Second International Symposium, SAIV 2025, Zagreb, Croatia, July 21–22, 2025, Proceedings*, Berlin, Heidelberg, 2025. Springer-Verlag.

[7] *AI Verification: Third International Symposium, SAIV 2026, Lisbon, Portugal, July 24–25, 2026, Proceedings*, Berlin, Heidelberg, 2026. Springer-Verlag.

[8] *Computer Aided Verification: 38th International Conference, CAV 2026, Lisbon, Portugal, July 26–29, 2026, Proceedings, Part I*, Berlin, Heidelberg, 2026. Springer-Verlag.

[9] Gavin Abercrombie and Verena Rieser. Risk-graded safety for handling medical queries in conversational ai. In *Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing*, pages 234–243, 2022.

[10] Matthias Althoff. An introduction to CORA 2015. In *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems (ARCH)*, pages 120–151, 2015.

[11] Matthias Althoff, Niklas Kochdumper, Tobias Ladner, Maximilian Perschl, and Mark Wetzlinger. CORA manual. *Technical University of Munich*, 2025.

[12] Ali ArjomandBigdeli, Andrew Mata, and Stanley Bak. *Verification of Neural Network Control Systems in Continuous Time*, page 100–115. Springer Nature Switzerland, 2024.

[13] Stanley Bak. Execution-guided overapproximation (ego) for improving scalability of neural network verification, 2020.

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

[15] Stanley Bak. Verifying neural networks with vibe coding. https://stanleybak.com/vibecheck/, July 2026. Accessed: 2026-10-03.

[16] Stanley Bak, Changliu Liu, and Taylor Johnson. The second international verification of neural networks competition (vnn-comp 2021): Summary and results, 2021.

[17] Stanley Bak, Hoang-Dung Tran, Kerianne Hobbs, and Taylor T. Johnson. Improved geometric path enumeration for verifying ReLU neural networks. In *32nd International Conference on Computer-Aided Verification (CAV)*, July 2020.

[18] Stanley Bak and Doug Wehbe. vibecheck-nn: Vibe-coded neural network verification tool. https://github.com/stanleybak/vibecheck-nn, 2026. GitHub repository, accessed: 2026-10-03.

[19] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB standard: Version 2.7. Technical report, Department of Computer Science, The University of Iowa, 2025.

[20] Timothy W Bickmore, Ha Trinh, Stefan Olafsson, Teresa K O'Leary, Reza Asadi, Nathaniel M Rickles, and Ricardo Cruz. Patient and consumer safety risks when using conversational assistants for medical information: an observational study of siri, alexa, and google assistant. *Journal of medical Internet research*, 20(9):e11510, 2018.

[21] Gregory Bonaert, Dimitar I Dimitrov, Maximilian Baader, and Martin Vechev. Fast and precise certification of transformers. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pages 466–481, 2021.

[22] Christopher Brix, Stanley Bak, Taylor T. Johnson, and Haoze Wu. The fifth international verification of neural networks competition (VNN-COMP 2024): Summary and results. 2024.

[23] Christopher Brix, Stanley Bak, Changliu Liu, and Taylor T. Johnson. The fourth international verification of neural networks competition (vnn-comp 2023): Summary and results. 2023.

[24] Christopher Brix, Mark Niklas Müller, Stanley Bak, Taylor T. Johnson, and Changliu Liu. First three years of the international verification of neural networks competition (vnn-comp), 2023.

[25] Rudy Bunel, Ilker Turkaslan, Philip HS Torr, Pushmeet Kohli, and M Pawan Kumar. A unified view of piecewise linear neural network verification. *Advances in Neural Information Processing Systems*, 2018.

[26] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A Limited Memory Algorithm for Bound Constrained Optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[27] Marco Casadio, Luca Arnaboldi, Matthew L Daggitt, Omri Isac, Tanvi Dinkar, Daniel Kienitz, Verena Rieser, and Ekaterina Komendantskaya. Antonio: Towards a systematic method for generating nlp benchmarks for verification. In *Proceedings of the 6th Workshop on Formal*, volume 16, pages 59–70, 2023.

[28] Marco Casadio, Tanvi Dinkar, Ekaterina Komendantskaya, Luca Arnaboldi, Omri Isac, Matthew L Daggitt, Guy Katz, Verena Rieser, and Oliver Lemon. Nlp verification: Towards a general methodology for certifying robustness. *arXiv preprint arXiv:2403.10144*, 2024.

[29] Marco Casadio, Ekaterina Komendantskaya, Matthew L. Daggitt, Wen Kokke, Guy Katz, Guy Amir, and Idan Refaeli. Neural network robustness as a verification property: A principled case study. In Sharon Shoham and Yakir Vizel, editors, *Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part I*, volume 13371 of *Lecture Notes in Computer Science*, pages 219–231. Springer, 2022.

[30] Minhao Cheng, Simranjit Singh, Patrick Chen, Pin-Yu Chen, Sijia Liu, and Cho-Jui Hsieh. Sign-opt: A query-efficient hard-label adversarial attack, 2020.

[31] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023.

[32] Lucas C. Cordeiro, Matthew L. Daggitt, Julien Girard-Satabin, Omri Isac, Taylor T. Johnson, Guy Katz, Ekaterina Komendantskaya, Augustin Lemesle, Edoardo Manino, Artjoms Šinkarovs, and Haoze Wu. Neural Network Verification is a Programming Language Challenge. In Viktor Vafeiadis, editor, *Programming Languages and Systems*, pages 206–235. Springer Nature Switzerland, 2025.

[33] Sarthak Das. Sobolbox: Boxed refinement of sobol sequence samples for neural network verification (competition contribution). In Mirco Giacobbe and Anna Lukina, editors, *AI Verification*, pages 272–277, Cham, 2026. Springer Nature Switzerland.

[34] Alessandro De Palma, Rudy Bunel, Krishnamurthy (Dj) Dvijotham, M. Pawan Kumar, Robert Stanforth, and Alessio Lomuscio. Expressive losses for verified robustness via convex combinations. In *Proceedings of the 12th International Conference on Learning Representations (ICLR 2024)*, pages 1–28, 2024.

[35] Stefano Demarchi, Dario Guidotti, Luca Pulina, and Armando Tacchella. Supporting standardization of neural networks verification with vnnlib and coconet. In Nina Narodytska, Guy Amir, Guy Katz, and Omri Isac, editors, *Proceedings of the 6th Workshop on Formal Methods for ML-Enabled Autonomous Systems*, volume 16 of *Kalpa Publications in Computing*, pages 47–58. EasyChair, 2023.

[36] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.

[37] Tianyu Du, Shouling Ji, Lujia Shen, Yao Zhang, Jinfeng Li, Jie Shi, Chengfang Fang, Jianwei Yin, Raheem Beyah, and Ting Wang. Cert-rnn: Towards certifying the robustness of recurrent neural networks. *CCS*, 21(2021):15–19, 2021.

[38] Hai Duong, Thanh Le, Lam Nguyen, and ThanhVu Nguyen. Verifying structural robustness of deep neural network. *Proceedings of the ACM on Software Engineering*, 3(FSE):1492–1514, 2026.

[39] Hai Duong, Thanh Le, and ThanhVu Nguyen. Clad: Constrained abstract domain for neural network verification. *arXiv preprint arXiv:2609.34628*, 2026.

[40] Hai Duong, Thanh Le, and ThanhVu Nguyen. Zonogpt: Towards an abstract domain for verifying large gpt models. *arXiv preprint arXiv:2609.34457*, 2026.

[41] Hai Duong, Lam Nguyen, Thanh Le, and ThanhVu Nguyen. Verifying neural network robustness with dual perturbations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 27916–27925, 2026.

[42] Hai Duong and ThanhVu Nguyen. Neuralsat: Scaling constraint solving for dnn verification (competition contribution). In *International Symposium on AI Verification*, pages 253–259. Springer, 2025.

[43] Hai Duong, ThanhVu Nguyen, and Matthew Dwyer. A dpll (t) framework for verifying deep neural networks. *arXiv preprint arXiv:2307.10266*, 2023.

[44] Hai Duong, ThanhVu Nguyen, and Matthew Dwyer. Generating and checking dnn verification proofs. *Advances in Neural Information Processing Systems*, 38:65887–65909, 2026.

[45] Hai Duong, ThanhVu Nguyen, and Matthew B Dwyer. Neuralsat: A high-performance verification tool for deep neural networks. In *International Conference on Computer Aided Verification*, pages 409–423. Springer, 2025.

[46] Hai Duong, David Shriver, ThanhVu Nguyen, and Matthew Dwyer. Compositional neural network verification via assume-guarantee reasoning. *Advances in Neural Information Processing Systems*, 38:64158–64182, 2026.

[47] Hai Duong, Le Thanh, and ThanhVu Nguyen. Verifying neural networks with reinforcement learning. *accepted at Advances in Neural Information Processing Systems*, 2026.

[48] Hai Duong, Dong Xu, ThanhVu Nguyen, and Matthew B Dwyer. Harnessing neuron stability to improve dnn verification. *Proceedings of the ACM on Software Engineering*, 1(FSE):859–881, 2024.

[49] Serge Durand, Augustin Lemesle, Zakaria Chihani, Caterina Urban, and François Terrier. Reciph: Relational coefficients for input partitioning heuristic. In *1st Workshop on Formal Verification of Machine Learning (WFVML 2022)*, 2022.

[50] EASA and Collins Aerospace. Formal Methods use for Learning Assurance (ForMuLA). Technical report, April 2023.

[51] Yizhak Yisrael Elboher, Omri Isac, Guy Katz, Tobias Ladner, and Haoze Wu. Abstraction-based proof production in formal verification of neural networks. In *International Symposium on AI Verification*, pages 203–220. Springer, 2025.

[52] James Ferlez, Haitham Khedr, and Yasser Shoukry. Fast BATLLNN: fast box analysis of two-level lattice neural networks. In Ezio Bartocci and Sylvie Putot, editors, *HSCC '22: 25th ACM International Conference on Hybrid Systems: Computation and Control, Milan, Italy, May 4 - 6, 2022*, pages 23:1–23:11. ACM, 2022.

[53] Kota Fukuda, Guanqin Zhang, Zhenya Zhang, Yulei Sui, and Jianjun Zhao. Adaptive branch-and-bound tree exploration for neural network verification. In *2025 Design, Automation & Test in Europe Conference (DATE)*, pages 1–7. IEEE, 2025.

[54] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018.

[55] David Gros, Yu Li, and Zhou Yu. The rua-robot dataset: Helping avoid chatbot deception

by detecting user questions about human or non-human identity. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6999–7013, 2021.

[56] Xingwu Guo, Wenjie Wan, Zhaodi Zhang, Min Zhang, Fu Song, and Xuejun Wen. Eager falsification for accelerating robustness verification of deep neural networks. In *2021 IEEE 32nd Int. Symp. on Software Reliability Engineering (ISSRE)*, pages 345–356. IEEE, 2021.

[57] Navid Hashemi, Samuel Sasaki · Ipek Oguz, Meiyi Ma, and Taylor Johnsonk. Scaling data-driven probabilistic robustness analysis for semantic segmentation neural networks. *NeurIPS*, 2025.

[58] Hanjiang Hu, Changliu Liu, and Ding Zhao. Robustness verification for perception models against camera motion perturbations. In *International conference on machine learning (ICML) Workshop on Formal Verification of Machine Learning (WFVML)*, 2023.

[59] Hanjiang Hu, Zuxin Liu, Linyi Li, Jiacheng Zhu, and Ding Zhao. Robustness certification of visual perception models via camera motion smoothing. In *Conference on Robot Learning*, pages 1309–1320. PMLR, 2022.

[60] Hanjiang Hu, Zuxin Liu, Linyi Li, Jiacheng Zhu, and Ding Zhao. Pixel-wise smoothing for certified robustness against camera motion perturbations. In *International Conference on Artificial Intelligence and Statistics*, pages 217–225. PMLR, 2024.

[61] Po-Sen Huang, Robert Stanforth, Johannes Welbl, Chris Dyer, Dani Yogatama, Sven Gowal, Krishnamurthy Dvijotham, and Pushmeet Kohli. Achieving verified robustness to symbol substitutions via interval bound propagation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4083–4093, 2019.

[62] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized Neural Networks. *Advances in Neural Information Processing Systems*, 29, 2016.

[63] Omri Isac, Clark W. Barrett, Min Zhang, and Guy Katz. Neural network verification with proof production. In Alberto Griggio and Neha Rungta, editors, *22nd Formal Methods in Computer-Aided Design, FMCAD 2022, Trento, Italy, October 17-21, 2022*, pages 38–48. IEEE, 2022.

[64] Robin Jia, Aditi Raghunathan, Kerem Göksel, and Percy Liang. Certified robustness to adversarial word substitutions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4129–4142, 2019.

[65] Stephen Joe and Frances Y. Kuo. Constructing sobol sequences with better two-dimensional projections. *SIAM Journal on Scientific Computing*, 30(5):2635–2654, 2008.

[66] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[67] Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar 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.

[68] Sydney M. Katz, Anthony L. Corso, Christopher A. Strong, and Mykel J. Kochenderfer. Verification of image-based neural network controllers using generative models, 2021.

[69] Konstantin Kaulen and Holger H Hoos. CTRAIN - a training library for certifiably robust neural networks. In *Proceedings of the 8th International Symposium on AI Verification (SAIV 2025)*, pages 1–12, 2025.

[70] Konstantin Kaulen and Holger H. Hoos. Automated algorithm configuration of α, β-CROWN. In *Proceedings of the 9th International Symposium on AI Verification (SAIV 2026)*, pages 232–253, 2026.

[71] Konstantin Kaulen, Matthias König, and Holger H Hoos. Dynamic algorithm termination for

branch-and-bound-based neural network verification. In *Proceedings of the 39th AAAI Conference on Artificial Intelligence (AAAI-25)*, volume 39, pages 27356–27364, 2025.

[72] Konstantin Kaulen, Tobias Ladner, Stanley Bak, Christopher Brix, Hai Duong, Thomas Flinkow, Taylor T. Johnson, Lukas Koller, Edoardo Manino, ThanhVu H Nguyen, et al. The 6th international verification of neural networks competition (VNN-COMP 2025): Summary and results. *arXiv preprint arXiv:2512.19007*, 2025.

[73] Konstantin Kaulen, Hadar Shavit, and Holger H. Hoos. Rethinking evaluation paradigms in ibp-based certified training. In *Proceedings of the 43rd International Conference on Machine Learning (ICML 2026)*, pages 1–38, 2026.

[74] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, 1995.

[75] Dmitrii Kirov and Simone Fulvio Rollini. Benchmark: remaining useful life predictor for aircraft equipment. In *International Conference on Bridging the Gap between AI and Reality*, pages 299–304. Springer, 2023.

[76] Dmitrii Kirov, Simone Fulvio Rollini, Luigi Di Guglielmo, and Darren Cofer. Formal verification of a neural network based prognostics system for aircraft equipment. In *International Conference on Bridging the Gap between AI and Reality*, pages 225–240. Springer, 2023.

[77] Ching-Yun Ko, Zhaoyang Lyu, Lily Weng, Luca Daniel, Ngai Wong, and Dahua Lin. Popqorn: Quantifying robustness of recurrent neural networks. In *International Conference on Machine Learning*, pages 3468–3477. PMLR, 2019.

[78] Niklas Kochdumper, Christian Schilling, Matthias Althoff, and Stanley Bak. Open- and closed-loop neural network verification using polynomial zonotopes. In *NASA Formal Methods*, pages 16–36, 2023.

[79] Lukas Koller, Tobias Ladner, and Matthias Althoff. Set-based training for neural network verification. *Transactions on Machine Learning Research (TMLR)*, 2025.

[80] Lukas Koller, Tobias Ladner, and Matthias Althoff. Out of the shadows: Exploring a latent space for neural network verification. *Proc. of the International Conference on Learning Representations (ICLR)*, 2026.

[81] Lukas Koller, Tobias Ladner, Benedikt Kellner, and Matthias Althoff. CORA: Verifying neural networks in a shared latent space (competition contribution). *Proc. of the International Symposium on AI Verification (SAIV)*, 2026.

[82] Mauritz Kop. Eu artificial intelligence act: The european approach to ai, 2021.

[83] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, 2018.

[84] Tobias Ladner and Matthias Althoff. Automatic abstraction refinement in neural network verification using sensitivity analysis. In *Proc. of the International Conference on Hybrid Systems: Computation and Control (HSCC)*, pages 1–13, 2023.

[85] Augustin Lemesle, Julien Lehmann, Tristan Le Gall, and Zakaria Chihani. Verifying neural networks with pyrat. In Hakjoo Oh and Yulei Sui, editors, *Static Analysis*, pages 11–33, Cham, 2026. Springer Nature Switzerland.

[86] Augustin Lemesle, Julien Lehmann, Tristan Le Gall, and Zakaria Chihani. Pyrat: Verifying neural networks with abstract interpretation (competition contribution). In Mirco Giacobbe and Anna Lukina, editors, *AI Verification*, pages 266–271, Cham, 2026. Springer Nature Switzerland.

[87] Augustin Lemesle, Julien Lehmann, and Le Gall Tristan. Neural Network Verification with PyRAT. *arXiv preprint arXiv:2410.23903*, 2024.

[88] Linhan Li and ThanhVu Nguyen. Destabilizing neurons to generate challenging neural network verification benchmarks. In *Proceedings of the 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ASE '25, 2025. To appear.

[89] Shuyi Lin, Haoyu He, Tianhao Wei, Kaidi Xu, Huan Zhang, Gagandeep Singh, Changliu Liu, and Cheng Tan. NN4SysBench: Characterizing neural network verification for computer systems. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, volume 37, pages 91390–91404. Curran Associates, Inc., 2024.

[90] Kaijie Liu and Yulei Sui. Detecting unsoundness in neural network verifiers via concrete–abstract consistency. In *Proceedings of the 3rd ACM International Conference on AI-Powered Software*, pages 110–119, 2026.

[91] Diego Manzanas Lopez, Sung Woo Choi, Hoang-Dung Tran, and Taylor T. Johnson. NNV 2.0: The neural network verification tool. In *35th International Conference on Computer-Aided Verification (CAV)*, July 2023.

[92] Diego Manzanas Lopez, Samuel Sasaki, and Taylor T. Johnson. Nnv: A star set reachability approach (competition contribution). In Mirco Giacobbe and Anna Lukina, editors, *AI Verification*, pages 260–265, Cham, 2026. Springer Nature Switzerland.

[93] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[94] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

[95] Diego Manzanas Lopez, Patrick Musau, Nathaniel Hamilton, and Taylor Johnson. Reachability analysis of a general class of neural ordinary differential equation. In *Proceedings of the 20th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2022), Co-Located with CONCUR, FMICS, and QEST as part of CONFEST 2022.*, Warsaw, Poland, September 2022.

[96] Milad Moradi and Matthias Samwald. Evaluating the robustness of neural language models to input perturbations. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 1558–1570, 2021.

[97] Mark Niklas Müller, Christopher Brix, Stanley Bak, Changliu Liu, and Taylor T Johnson. The third international verification of neural networks competition (vnn-comp 2022): Summary and results. *arXiv preprint arXiv:2212.10376*, 2022.

[98] Andreea Postovan and Mădălina Eraşcu. Architecturing binarized neural networks for traffic sign recognition. *arXiv preprint arXiv:2303.15005*, 2023.

[99] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China, November 2019. Association for Computational Linguistics.

[100] Jiawei Ren, Guanqin Zhang, Zhenya Zhang, and Yulei Sui. Mining verdict boundaries for neural network verification. In *International Symposium on Formal Methods*, pages 67–88. Springer, 2026.

[101] Ann Roy, Allen Antony, Andrea Gimelli, and Matthew L. Daggitt. VNN-LIB 2.0: Rigorous Foundations for Neural Network Verification, May 2026.

[102] Samuel Sasaki, Diego Manzanas Lopez, Preston K. Robinette, and Taylor T. Johnson. Robustness verification of video classification neural networks. In *2025 IEEE/ACM 13th International Conference on Formal Methods in Software Engineering (FormaliSE)*, pages 22–33, 2025.

[103] Samuel Sasaki, Ben Wooding, Hanchen David Wang, Anne M Tumlin, Meiyi Ma, and Taylor T Johnson. n2v: Neural network verification in python (competition contribution). In *International Symposium on AI Verification*, pages 394–400. Springer, 2026.

[104] Zhouxing Shi, Cho-Jui Hsieh, and Huan Zhang. Certified training with branch-and-bound: A case study on lyapunov-stable neural control. *arXiv preprint arXiv:2411.18235*, 2024.

[105] Zhouxing Shi, Qirui Jin, Zico Kolter, Suman Jana, Cho-Jui Hsieh, and Huan Zhang. Neural network verification with branch-and-bound for general nonlinearities. *arXiv preprint arXiv:2405.21063*, 2024.

[106] Zhouxing Shi, Yihan Wang, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Fast certified robust training with short warmup. *Advances in Neural Information Processing Systems*, 34:18335–18349, 2021.

[107] Zhouxing Shi, Huan Zhang, Kai-Wei Chang, Minlie Huang, and Cho-Jui Hsieh. Robustness verification for transformers. In *International Conference on Learning Representations*, 2019.

[108] Zhouxing Shi, Huan Zhang, Kai-Wei Chang, Minlie Huang, and Cho-Jui Hsieh. Robustness verification for transformers, 2020.

[109] David Shriver, Sebastian Elbaum, and Matthew 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, 2021.

[110] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[111] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 3(POPL):41:1–41:30, 2019.

[112] Ilya M. Sobol'. On the Distribution of Points in a Cube and the Approximate Evaluation of Integrals. *Ussr Computational Mathematics and Mathematical Physics*, 7:86–112, 1967.

[113] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing Properties of Neural Networks. *arXiv preprint arXiv:1312.6199*, 2013.

[114] Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *ICLR*, 2019.

[115] H. D. Tran, N. Pal, D. Lopez, P. Musau, X. Yang, W. Xiang L. Nguyen, S. Bak, , and T. T. Johnson. Verification of piecewise deep neural networks: A star set approach with zonotope pre-filter. *Formal aspects of computing*, 2021.

[116] Hoang-Dung Tran, Stanley Bak, Weiming Xiang, and Taylor T. Johnson. Verification of deep convolutional neural networks using imagestars. In *32nd International Conference on Computer-Aided Verification (CAV)*. Springer, July 2020.

[117] Hoang-Dung Tran, Feiyang Cei, Diego Manzanas Lopez, Taylor T. Johnson, and Xenofon Koutsoukos. Safety verification of cyber-physical systems with reinforcement learning control. In *ACM SIGBED International Conference on Embedded Software (EMSOFT'19)*. ACM, October 2019.

[118] Hoang Dung Tran, SungWoo Choi, Tomoya Yamaguchi, Bardh Hoxha, and Danil Prokhorov. Verification of recurrent neural networks using star reachability. In *The 26th ACM International Conference on Hybrid Systems: Computation and Control (HSCC)*, May 2023.

[119] Hoang-Dung Tran, Patrick Musau, Diego Manzanas Lopez, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T. Johnson. Star-based reachability analysis for deep neural networks. In *23rd International Symposium on Formal Methods (FM'19)*. Springer International Publishing, October 2019.

[120] Hoang-Dung Tran, Neelanjana Pal, Patrick Musau, Xiaodong Yang, Nathaniel P. Hamilton, Diego Manzanas Lopez, Stanley Bak, and Taylor T. Johnson. Robustness verification of semantic segmentation neural networks using relaxed reachability. In *33rd International Conference on Computer-Aided Verification (CAV)*. Springer, July 2021.

[121] Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T. Johnson. NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *32nd International Conference on Computer-Aided Verification (CAV)*, July 2020.

[122] Anne M Tumlin, Diego Manzanas Lopez, Preston Robinette, Yuying Zhao, Tyler Derr, and Taylor T Johnson. Fairnnv: The neural network verification tool for certifying fairness. ICAIF

'24, page 36–44, New York, NY, USA, 2024. Association for Computing Machinery.

[123] Anne M Tumlin, Samuel Sasaki, Ben Wooding, Diego Manzanas Lopez, Muhammad Usama Zubair, Navid Hashemi, Hongchao Zhang, Waseem Abbas, Ipek Oguz, Meiyi Ma, and Taylor T. Johnson. NNV3: Expanding neural network verification to new architectures and domains. *arXiv preprint arXiv:2609.30050*, 2026.

[124] Anne M Tumlin, Ben Wooding, Zhenxuan Shao, Diego Manzanas Lopez, Tyler Derr, and Taylor T Johnson. Reachability-based formal verification of graph neural networks with node and edge features. In *International Symposium on AI Verification*, pages 271–298. Springer, 2026.

[125] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[126] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and Zico Kolter. Beta-CROWN: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. *arXiv preprint arXiv:2103.06624*, 2021.

[127] Yibin Wang, Yichen Yang, Di He, and Kun He. Robustness-aware word embedding improves certified robustness to adversarial word substitutions. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 673–687, 2023.

[128] Johannes Welbl, Po-Sen Huang, Robert Stanforth, Sven Gowal, Krishnamurthy Dj Dvijotham, Martin Szummer, and Pushmeet Kohli. Towards verified robustness under text deletion interventions. 2020.

[129] Manuel Wendl, Lukas Koller, Tobias Ladner, and Matthias Althoff. Training verifiably robust agents using set-based reinforcement learning. *Transactions on Machine Learning Research (TMLR)*, 2026.

[130] Haoze Wu, Teruhiro Tagomori, Alexander Robey, Fengjun Yang, Nikolai Matni, George Pappas, Hamed Hassani, Corina Pasareanu, and Clark Barrett. Toward certified robustness against real-world distribution shifts. *arXiv preprint arXiv:2206.03669*, 2022.

[131] Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. Automatic perturbation analysis for scalable certified robustness and beyond. *Advances in Neural Information Processing Systems*, 33, 2020.

[132] Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. Fast and Complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In *International Conference on Learning Representations*, 2021.

[133] Lujie Yang, Hongkai Dai, Zhouxing Shi, Cho-Jui Hsieh, Russ Tedrake, and Huan Zhang. Lyapunov-stable neural control for state and output feedback: A novel formulation. In *Forty-first International Conference on Machine Learning*, 2024.

[134] Yujie Yang, Hanjiang Hu, Tianhao Wei, Shengbo Eben Li, and Changliu Liu. Scalable synthesis of formally verified neural value function for hamilton-jacobi reachability analysis. *Journal of Artificial Intelligence Research*, 83, 2025.

[135] Guanqin Zhang, Kota Fukuda, Zhenya Zhang, HMN Bandara, Shiping Chen, Jianjun Zhao, and Yulei Sui. Efficient neural network verification via order leading exploration of branch-and-bound trees. *arXiv preprint arXiv:2507.17453*, 2025.

[136] Guanqin Zhang and Yulei Sui. Tensor-based batch fuzzing with adaptive perturbation scaling for deep neural networks. *arXiv preprint arXiv:2606.25239*, 2026.

[137] Huan Zhang*, Shiqi Wang*, Kaidi Xu*, Linyi Li, Bo Li, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. General cutting planes for bound-propagation-based neural network verification. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[138] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. *Advances in Neural Information Processing Systems*, 31:4939–4948, 2018.

[139] Jianming Zhang, Wei Wang, Chaoquan Lu, Jin Wang, and Arun Kumar Sangaiah. Lightweight

Deep Network for Traffic Sign Classification. *Annals of Telecommunications*, 75:369–379, 2020.

[140] Yuhao Zhang, Aws Albarghouthi, and Loris D'Antoni. Certified robustness to programmable transformations in lstms. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 1068–1083, 2021.

[141] Duo Zhou, Christopher Brix, Grani A Hanasusanto, and Huan Zhang. Scalable neural network verification with branch-and-bound inferred cutting planes. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[142] Xingjian Zhou, Hongji Xu, Andy Xu, Zhouxing Shi, Cho-Jui Hsieh, and Huan Zhang. Testing neural network verifiers: A soundness benchmark with hidden counterexamples. *arXiv preprint arXiv:2412.03154*, 2024.

[143] Muhammad Usama Zubair, Taylor T. Johnson, Kanad Basu, and Waseem Abbas. Verification of neural network robustness against weight perturbations using star sets. In *2025 IEEE Conference on Artificial Intelligence (CAI)*, pages 637–642, 2025.

[144] Muhammad Usama Zubair, Taylor T. Johnson, Kanad Basu, and Waseem Abbas. Modelstar: Reachability analysis-based safety verification of neural networks against model perturbations. *J. Artif. Int. Res.*, 85, April 2026.