Marabou 2.0: A Versatile Formal Analyzer of Neural Networks

Haoze Wu[1](✉), Omri Isac[2], Aleksandar Zeljić[1], Teruhiro Tagomori[1,3], Matthew Daggitt[4], Wen Kokke[5], Idan Refaeli[2], Guy Amir[2], Kyle Julian[1], Shahaf Bassan[2], Pei Huang[1], Ori Lahav[2], Min Wu[1], Min Zhang[6], Ekaterina Komendantskaya[4], Guy Katz[2], and Clark Barrett[1]

[1] Stanford University, Stanford, USA
haozewu@stanford.edu
[2] The Hebrew University of Jerusalem, Jerusalem, Israel
[3] NRI Secure, Palo Alto, USA
[4] Heriot-Watt University, Edinburgh, UK
[5] University of Strathclyde, Glasgow, UK
[6] East China Normal University, Shanghai, China

Abstract. This paper serves as a comprehensive system description of version 2.0 of the Marabou framework for formal analysis of neural networks. We discuss the tool's architectural design and highlight the major features and components introduced since its initial release.

1 Introduction

With the increasing pervasiveness of deep neural networks (DNNs), the formal analysis of DNNs has become a burgeoning research field within the formal methods community. Multiple DNN reasoners have been proposed in the past few years, including α-β-CROWN [56,65,69], ERAN [45–47], Marabou [32], MN-BaB [16], NNV [35,51], nnenum [4], VeriNet [24,25], and many others.

We focus here on the Marabou [32] tool, which has been used by the research community in a wide range of formal DNN reasoning applications (e.g., [9,12,17, 18,22,26,34,37,49,54,64,66], inter alia). Initially, Marabou was introduced as a from-scratch re-implementation of the Reluplex [31] decision procedure, with a native linear programming engine and limited support for DNN-level reasoning. Over the years, fundamental changes have been made to the tool, not only from an algorithmic perspective but also to its engineering and implementation.

This paper introduces version 2.0 of Marabou. Compared to its predecessor, Marabou 2.0: (i) employs a new build/test system; (ii) has an optimized core system architecture; (iii) runs an improved decision procedure and abstract interpretation techniques; (iv) handles a wider range of activation functions;(v) supports proof production; (vi) supports additional input formats; and (vii) contains a more powerful Python API. Due to these changes, the original system description [32] no longer gives an accurate account of the tool's current capabilities. Our goal in this paper is to close this gap and provide a comprehensive

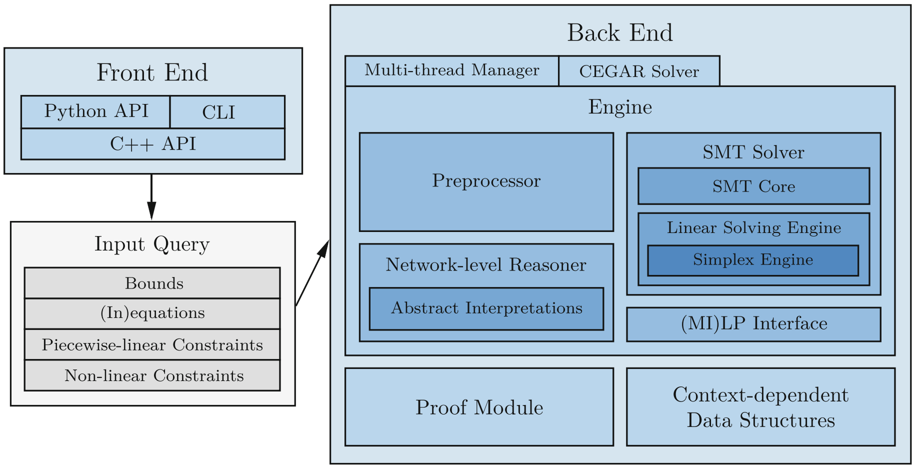

Fig. 1. High-level overview of Marabou 2.0's system architecture.

description of the current Marabou system. We highlight the major features introduced since the initial version, describe a few of its many recent uses, and report on its performance, as demonstrated by the VNN-COMP'23 results and additional runtime comparisons against an early version of Marabou.

2 Architecture and Core Components

In this section, we discuss the core components of Marabou 2.0. An overview of its system architecture is given in Fig. 1. At a high level, Marabou performs satisfiability checking on a set of linear and non-linear constraints, supplied through one of the front-end interfaces. The constraints typically represent a verification query over a neural network and are stored in an *InputQuery* object. We distinguish variable bounds from other linear constraints, and piecewise-linear constraints (which can be reduced to linear constraints via case analysis) from more general, non-linear constraints.

Variables are represented as consecutive indices starting from 0. (In)equations are represented as *Equation* objects. Piecewise-linear constraints are represented by objects of classes that inherit from the *PiecewiseLinearConstraint* abstract class. The abstract class defines the key interface methods that are implemented in each sub-class. This way, all piecewise-linear constraints are handled uniformly in the back end. Similarly, each other type of non-linear constraint is implemented as a sub-class of the new *NonlinearConstraint* abstract class. Initially, Marabou only supported the ReLU and Max constraints. In Marabou 2.0, over ten types of non-linear constraints (listed in the extended version of the paper [61]) are supported.

2.1 Engine

The centerpiece of Marabou is called the Engine, which reasons about the satisfiability of the input query. The engine consists of several components: the Preprocessor, which performs rewrites and simplifications; the Network-level Reasoner, which maintains the network architecture and performs all analyses that require this knowledge; the SMT Solver, which houses complete decision procedures for sets of linear and piecewise-linear constraints; and the (MI)LP Interface, which manages interactions with external (MI)LP solvers for certain optional solving modes as explained below.

Two additional modules are built on top of the Engine. The Multi-thread Manager spawns multiple Engine instances to take advantage of multiple processors. The CEGAR Solver performs incremental linearization [13,62] for non-linear constraints that cannot be precisely handled by the SMT Solver.

Preprocessor. Every verification query first goes through multiple preprocessing passes, which *normalize*, *simplify*, and *rewrite* the query. One new normalizing pass introduces auxiliary variables and entailed linear constraints for each of the piecewise-linear constraints, so that case splits on the piecewise-linear constraints can be represented as bound updates and consequently do not require adding new equations.[1] This accelerates the underlying Simplex engine, as explained in the SMT Solver section below. Another significant preprocessing pass involves iterative bound propagation over all constraints. In this process, piecewise linear constraints might collapse into linear constraints and be removed. This pass was present in Marabou 1.0, but could become a runtime bottleneck; whereas Marabou 2.0 employs a data structure optimization that leads to a ∼60x speed up. Finally, the preprocessor merges any variables discovered to be equal to each other and also eliminates any constant variables. This results in updates to the variable indices, and therefore a mapping from old indices to new ones needs to be maintained for retrieving satisfying assignments.

SMT Solver. The SMT Solver module implements a sound and complete, lazy-DPLL(T)-based procedure for deciding the satisfiability of a set of linear and piecewise-linear constraints. It performs case analysis on the piecewise-linear constraints and, at each search state, employs a specialized procedure to iteratively search for an assignment satisfying both the linear and non-linear constraints.

Presently, the DeepSoI procedure [58] has replaced the Reluplex procedure [31,32] as Marabou's default procedure to run at each search state. The former provably converges to a satisfying assignment (if it exists) and empirically consistently outperforms the latter. DeepSoI extends the canonical

[1] For example, for a piece-wise linear constraint $y = \max(x_1, x_2)$, we would introduce $c_1 : y - x_1 = a_1 \wedge a_1 \geq 0 \wedge y - x_2 = a_2 \wedge a_2 \geq 0$, where a_1 and a_2 are fresh variables. This way, case splits on this constraint can be represented as $c_2 : a_1 \leq 0$ and $c_3 : a_2 \leq 0$, respectively. This preprocessing pass preserves satisfiability because the original constraint is equisatisfiable to $c_1 \wedge (c_2 \vee c_3)$.

sum-of-infeasibilities method in convex optimization [10], which determines the satisfiability of a set of linear constraints by minimizing a cost function that represents the total violation of the constraints by the current assignment. The constraints are satisfiable if and only if the optimal value is 0. Similarly, DeepSoI formulates a cost function that represents the total violation of the current piecewise-linear constraints and uses a convex solver to stochastically minimize the cost function with respect to the convex relaxation of the current constraints. In addition, DeepSoI also informs the branching heuristics of the SMT Core, which performs a case split on the piecewise-linear constraint with the largest impact (measured by the *pseudocost* metric [58]) on the cost function. The DeepSoI procedure is implemented for all supported piecewise-linear activation functions. The convex solver can be instantiated either with the native Simplex engine or with an external LP solver via the (MI)LP interface (detailed below). The latter can be more efficient but requires the use of external commercial solvers.

One optimization in Marabou 2.0's Simplex engine is that once the tableau has been initialized, it avoids introducing any new equations — a costly operation that requires re-computing the tableau from scratch. This is achieved by implementing case-splitting and backtracking as updates on variable bounds (as mentioned above), which only requires minimal updates to the tableau state. By our measure, this optimization reduces the runtime of the Simplex engine by over 50%. Moreover, the memory footprint of the solver is also drastically decreased, as the SMT Core no longer needs to save the entire tableau state during case-splitting (to be restored during backtracking).

Network-Level Reasoner. Over the past few years, numerous papers (e.g., [41,46,55,68,70], inter alia) have proposed abstract interpretation techniques that rely on network-level reasoning (e.g., propagating the input bounds layer by layer to tighten output bounds). These analyses can be viewed as a stand-alone, incomplete DNN verification procedure, or as in-processing bound tightening passes for the SMT Solver. Marabou 2.0 features a brand new *NetworkLevelReasoner* class that supports this type of analysis. The class maintains the neural network topology as a directed acyclic graph, where each node is a *Layer* object. The *Layer* class records key information such as weights, source layers, and mappings between neuron indices and variable indices. Currently, seven different analyses are implemented:[i] 1.interval bound propagation [20]; 2. symbolic bound propagation [55]; 3. DeepPoly/CROWN analysis [46,70]; 4. LP-based bound tightening [50]; 5. Forward-backward analysis [59]; 6.MILP-based bound tightening [50]; and 7. iterative propagation [57]. Analyses 2–7 are implemented in a parallelizable manner, and analyses 4–7 require calls to an external LP solver. By default, the DeepPoly/CROWN analysis is performed. The Network-level Reasoner tightly interleaves with the SMT Solver: the network-level reasoning is executed any time a new search state is reached (with the most up-to-date variable bounds), and the derived bound tightenings are immediately fed back to the search procedure.

It is noteworthy that the user does not have to explicitly provide the neural network topology to enable network-level reasoning. Instead, the network architecture is *automatically inferred* from the given set of linear and non-linear constraints, via the *constructNetworkLevelReasoner* method in the *InputQuery* class. The Network-level Reasoner is only initialized if such inference is successful. Apart from the abstract interpretation passes, the Network-level Reasoner can also evaluate concrete inputs. This is used to implement the LP-based bound tightening optimization introduced by the NNV tool [51].

(MI)LP Interface. Marabou can now optionally be configured to invoke the Gurobi Optimizer [23], a state-of-the-art Mixed Integer Linear Programming (MILP) solver. The *GurobiWrapper* class contains methods to construct a MILP problem and invoke the solver. The *MILPEncoder* class is in charge of encoding the current set of linear and non-linear constraints as (MI)LP constraints. Piecewise-linear constraints can either be encoded precisely, or replaced with a convex relaxation, resulting in a linear program. For other non-linear constraints, only the latter option is available. The (MI)LP interface presently has three usages in the code base. Two have already been mentioned, i.e., in some of the abstract interpretation passes and optionally in the DeepSoI procedure. Additionally, when Marabou is compiled with Gurobi, a `--milp` mode is available, in which the Engine performs preprocessing and abstract interpretation passes, and then directly encodes the verification problem as a MILP problem to be solved by Gurobi. The mode is motivated by the observation that the performance of Gurobi and the SMT Solver can be complementary [48,58].

Multi-thread Manager. Parallelization is an important way to improve verification efficiency. Marabou supports two modes of parallelization, both managed by the new *MultiThreadManager* class: the *split-and-conquer* mode [57] and the *portfolio* mode. In the split-and-conquer mode, the original query is dynamically partitioned and re-partitioned into independent sub-queries, to be handled by idle workers. The partitioning strategy is implemented as a sub-class of the *QueryDivider* abstract class. Currently, two strategies are available: one partitions the intervals of the input variables; the other splits on piecewise linear constraints. By default, the former is used only when the input dimension is less than or equal to ten. In the portfolio mode, each worker solves the same query with a different random seed, which takes advantage of the stochastic nature of the DeepSoI procedure. Developing an interface to define richer kinds of portfolios is work in progress.

CEGAR Solver. While the DNN verification community has by and large focused on piecewise-linear activation functions, other classes of non-linear connections exist and are commonly used for certain architectures [27,53]. Apart from introducing support for non-linear constraints in the Preprocessor and the Network-level Reasoner, the latest Marabou version also incorporates a counter-example guided abstraction refinement (CEGAR) solving mode [62], based on incremental linearization [13] to enable more precise reasoning about non-linear constraints that are not piecewise linear. Currently, the CEGAR solver only supports Sigmoid and Tanh, but the module can be extended to handle other activation functions.

2.2 Context-Dependent Data-Structures

When performing a case split or backtracking to a previous search state, the SMT Core needs to save or restore information such as variable bounds and the phase status of each piecewise-linear constraint (e.g., is a ReLU currently active, inactive, or unfixed). To efficiently support these operations, Marabou 2.0 uses the notion of a context level (borrowed from the CVC4 SMT solver [6]), and stores the aforementioned information in *context-dependent data structures*. These data structures behave similarly to their standard counterparts, except that they are associated with a context level and *automatically* save and restore their state as the context increases or decreases. This major refactoring has greatly simplified the implementation of saving and restoring solver states and is an important milestone in an ongoing effort to integrate a full-blown Conflict-Driven Clause-Learning (CDCL) mechanism into Marabou.

2.3 Proof Module

A proof module has recently been introduced into Marabou, enabling it to optionally produce proof certificates after an unsatisfiable (UNSAT) [29] result. This is common practice in the SAT and SMT communities and is aimed at ensuring solver reliability. Marabou produces proof certificates based on a constructive variant of the Farkas lemma [52], which ensures the existence of a *proof vector* that witnesses the unsatisfiability of a linear program. Specifically, the *proof vector* corresponds to a linear equation that is violated by the variable bounds [29]. The full certificate of UNSAT is comprised of a *proof tree*, whose nodes represent the search states explored during the solving. Each node may contain a list of *lemmas* that are used as additional constraints in its descendent nodes; and each leaf node contains the proof vector for the unsatisfiability of the corresponding sub-query. The lemmas encapsulate some of the variable bounds, newly derived by the piecewise-linear constraints of the query, and require their own witnesses (i.e., proof vectors). The *BoundExplainer* class is responsible for constructing all proof vectors, for updating them during execution, and for appending them to the node. The proof tree itself is implemented using the *UnsatCertificateNode* class.

When the solver is run in proof-production mode, the Proof module closely tracks the steps of the SMT Solver module and constructs the proof tree on the fly: new nodes are added to the tree whenever a case split is performed; and a new proof vector is generated whenever a lemma is learned or UNSAT is derived for a sub-query. If the Engine concludes that the entire query is UNSAT, a proof checker (implemented as an instance of the *Checker* class) will be triggered to certify the proof tree. It does so by traversing the tree and certifying the

```
Q = Marabou.read_onnx("model.onnx")
X, Y = Q.inputVars[0], Q.outputVars[0]
Q.setLowerBound(X[0], 0.1)
Q.addInequality([Y[0], Y[1]], [1, -0.5], 0)
Q.solve()
```

(a) The base Python API

```
Q = Marabou.read_onnx("model.onnx")
X, Y = Q.inputVars[0], Q.outputVars[0]
Q.addConstraint(Var(X[0]) >= 0.1)
Q.addConstraint(Var(Y[0]) <= 0.5 * Var(Y[1]))
Q.solve()
```

(b) The "Pythonic" API

Fig. 2. Two ways to define the same verification query through the Python API.

correctness of the lemmas and the unsatisfiability of the leaf nodes. A formally verified and precise proof-checker is currently under development [14]. Note that, currently, proof production mode is only compatible with a subset of the features supported by Marabou. Adding support for the remaining features (e.g., for the parallel solving mode) is an ongoing endeavor.

2.4 Front End

Marabou provides interfaces to prepare input queries and invoke the back-end solver in multiple ways. The Marabou executable can be run on the command line, taking in network/property/query files in supported formats. The Python and C++ APIs support this functionality as well, but also contain methods to add arbitrary linear and (supported) non-linear constraints. In addition, a layer on top of the Python API was added to Marabou 2.0 which allows users to define constraints in a more *Pythonic* manner, resulting in more succinct code. For example, suppose one wants to check whether the first output of a network (stored in the ONNX format) can be less than or equal to half of its second output, when the first input is greater than or equal to 0.1. Figure 2a shows how to perform this check with the base Python API, while Fig. 2b exhibits the "Pythonic" API.

Typically, a query consists of the encoding of (one or several) neural networks and the encoding of a property on the network(s). To encode a neural network, the user has two options: 1) pass in a neural network file to be parsed by one of the neural network parsers; or 2) manually add constraints to encode the neural network. The main network format for Marabou 2.0 is now ONNX, towards which the neural network verification community is converging. The NNet format and the Tensorflow protobuf format are still supported but will likely be phased out in the long run. To encode the property on top of the neural network encoding, the user can 1) pass in a property file to be parsed by one of the property parsers; or 2) manually encode the property. Currently Marabou has two property parsers, one for a native property file format [32], and a new one for the VNN-LIB format, supporting the standardization effort of the community.

In addition to the aforementioned network and property file formats, Marabou also supports a native query file format that describes a set of linear and non-linear constraints. This can be dumped/parsed from all interfaces.

2.5 Availability, License, and Installation

Marabou is available under the permissive modified BSD open-source license, and runs on Linux and macOS machines. The tool can be built from scratch using CMake. Marabou is now also available on The Python Package Index (PyPI) and can be installed through `pip`. The latest version of Marabou is available at: https://github.com/NeuralNetworkVerification/Marabou. The artifact associated with this tool description is archived on Zenodo [60].

3 Highlighted Features and Applications

In terms of performance, Marabou is on par with state-of-the-art verification tools. In the latest VNN-COMP [11], Marabou won the second place overall, and scored the highest among all CPU-based verifiers. We summarize the main results in the extended version of the paper [61]. In this section, we focus on the usability aspect of Marabou, and highlight some of its recent applications — as well as the features that make them possible. We believe this diverse set of use cases (as well as the relevant scripts in the artifact [60]) serve as valuable examples, which will inspire new ways to apply the solver. More use cases can be found in the extended version of the paper [61]. A runtime evaluation of Marabou 2.0 against an early version appears in Sect. 4.

Verifying the Decima Job Scheduler. Recently, Graph Neural Networks (GNNs) have been used to schedule jobs over multi-user, distributed-computing clusters, achieving state-of-the-art job completion time [38]. However, concerns remain over whether GNN-based solutions satisfy expected cost-critical properties beyond performance. Marabou has been used to verify a well-known fairness property called *strategy-proofness* [59] for a high-profile, state-of-the-art GNN-based scheduler called Decima [38]. The verified property states that "a user cannot get their job scheduled earlier by misrepresenting their resource requirement." While it is challenging to represent a GNN directly in ONNX [21], Marabou's Python API makes it possible to manually encode Decima and the specification as a set of linear and non-linear constraints. From these constraints, the Network-level Reasoner is able to automatically infer a feed-forward structure with residual connections and then use it for the purpose of abstract interpretation. Notably, Marabou was able to handle the *original* Decima architecture, proving that the property holds on the vast majority of the examined job profiles but can indeed be violated in some cases.

Formal XAI. Despite their prevalence, DNNs are considered "black boxes", uninterpretable to humans. *Explainable AI* (XAI) aims to understand DNN decisions to enhance trust. Most XAI methods are heuristic-based and lack formal correctness guarantees [36,43,44], which can be problematic for critical, regulation-heavy systems. Recent work showed that Marabou can be utilized as a sub-routine in procedures designed for producing *formal and provable* explanations for DNNs [7,8,26,37,63]. For instance, it can be used in constructing formal *abductive explanations* [8,28], which are subsets of input features that are, by themselves, provably sufficient for determining the DNN's output. This approach has been successfully applied to large DNNs in the domains of computer vision [8,63], NLP [37], and DRL robotic navigation [7]. These studies highlight the potential of Marabou in tasks that go beyond formal verification.

Analyzing Learning-Based Robotic Systems. Deep Reinforcement Learning has extensive application in robotic planning and control. Marabou has been applied in these settings to analyze different safety and liveness properties [2,3,15,54]. For example, Amir et al. [2] used Marabou to detect infinite loops in a real-world robotic navigation platform. This was achieved by querying whether there exists a state to which the robot will always return within a finite number of steps k, effectively entering an infinite loop. A multi-step property like this can be conveniently encoded in Marabou, by (i) encoding k copies of the control policy; (ii) for each time-step t, encoding the system transition as constraints over the current state (input to the policy at t), the decided action (output of the policy at t), and the next state (input to the policy at $t + 1$); and (iii) encoding the "loop" constraint that the initial state (t_1) is equal to the final state (t_k). From this set of constraints, the Network-level Reasoner can infer the structure of and perform abstract interpretations over a *concatenated network*, where the input is the initial state and the output is the final state. Moreover, due to the low input dimension, the split-and-conquer mode in the Multi-thread Manager can be used to perform input-splitting, effectively searching for such loops in independent input regions in parallel. Notably, Marabou can detect loops in the system for agents trained using state-of-the-art RL algorithms, in cases where gradient/optimization-based approaches fail to find any. Loops detected this way have also been observed in the real world [1].

Proof Production for the ACAS-Xu Benchmarks. A well-studied set of benchmarks in DNN verification derives from an implementation of the ACAS-Xu airborne system for collision avoidance [30]. Using Marabou, we were able to produce certificates of unsatisfiability for these benchmarks for the first time. Marabou was able to produce certificates for 113 out of the 180 tested benchmarks, with only mild overhead incurred by proof generation and certification. The proof certificates contained over 1.46 million proof-tree leaves, of which more than 99.99% were certified by the native proof checker, while the remaining were certified by a trusted SMT solver. Additional details are provided in [29].

Specifications on Neural Activation Patterns. Properties of hidden neurons garner increasing interest [67], as they shed light on the internal decision-making process of the neural network. Gopinath et al. [19] observed that for a fixed neural network, certain *neuron activation patterns* (NAPs) empirically entail a fixed prediction. More recently, Geng et al. [18] formally verified (using Marabou) the aforementioned property, along with a variety of other properties related to NAPs. Specifications related to NAPs can be conveniently encoded in Marabou. For example, specifying that a certain ReLU is activated amounts to setting the lower bound of the variable corresponding to the ReLU input to 0, using the general constraint-encoding methods in the Python/C++ API. Constraints on internal neurons, as with other constraints, can be propagated by the Preprocessor and Network-level Reasoner to tighten bounds.

Robustness Against Semantically Meaningful Perturbations. Considering specifications of perception networks, there is an ongoing effort in the verification community to go beyond *adversarial robustness* [5,33,39,40,62]. Marabou has been used to verify robustness against semantically meaningful perturbations that can be analytically defined/abstracted as linear constraints on the neural network inputs (e.g., brightness, uniform haze) [42]. More recently, Marabou has also been successfully applied in a neural symbolic approach, where the correct network behavior is defined with respect to that of another network [62,64]. For example, Wu et al. [62] considered the specification that an image classifier's prediction does not change with respect to outputs of an image generative model trained to capture a complex distribution shift (e.g., change in weather condition). A property like this can be conveniently defined in Marabou by loading the classifier and the generator through the Python API and adding the relevant constraints on/between their input and output variables.

4 Runtime Evaluation

We measure the performance improvement in Marabou 2.0 by comparing it against an early Marabou version (git commit 1c1c66), which can handle ReLU and Max constraints and supports symbolic bound propagation [55]. We collected four benchmark sets from the applications described in Section 3: Alternating Loop [2], DeepCert [42], NAP [18,19], and VeriX [63]. There are 745 instances in total. Details about the benchmarks can be found in the extended version of the paper [61].

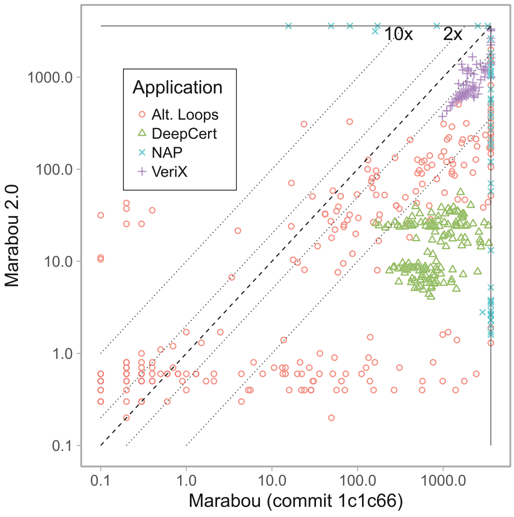

Fig. 3. Runtime performance of Marabou 2.0 and an early version of Marabou on four applications supported by both versions.

Figure 3 compares the runtime of the two Marabou versions on all the benchmarks with a 1 h CPU timeout. Each configuration was given 1 core and 8GB of memory. Note that Marabou 2.0 was not configured with external solvers in this experiment. We see that Marabou 2.0 is significantly more efficient for a vast majority of the instances. Upon closer examination, an at-least 2× speed-up is achieved on 428 instances and an at-least 10× speed-up is achieved on 263 instances. Moreover, Marabou 2.0 is also significantly more memory efficient, with a median peak usage of 57MB (versus 604MB with the old version). Solvers'performance on individual benchmarks is reported in the extended version of the paper [61].

5 Conclusion and Next Steps

We have summarized the current state of Marabou, a maturing formal analyzer for neural-network-enabled systems that is under active development. In its current form, Marabou is a versatile and user-friendly toolkit suitable for a wide range of formal analysis tasks. Moving forward, we plan to improve Marabou in several dimensions. Currently, we are actively integrating a CDCL mechanism in the SMT Solver module. Given that many applications involve repeated invocation of the solver on similar queries, we also plan to support incremental solving in the style of pushing and popping constraints, leveraging the newly introduced context-dependent data structures. In addition, adding GPU support (in the Network-level Reasoner) and handling other types of non-linear constraints are also on the development agenda for Marabou.

Acknowledgment. The work of Wu, Zeljić, Tagomori, Huang and Wu was partially supported by the NSF (grant number 2211505), by the BSF (grant number 2020250), a Ford Alliance Project (199909), the Stanford Center for AI Safety, and the Stanford Institute for Human-Centered Artificial Intelligence (HAI). The work of Daggit, Kokke and Komendantskaya was partially supported by the EPSRC grant EP/T026952/1, AISEC: AI Secure and Explainable by Construction. The work of Isac, Refaeli, Amir, Bassan, Lahav and Katz was partially funded by the ISF (grant number 3420/21), by the BSF (grant numbers 2021769 and 2020250), and by the European Union (ERC,VeriDeL, 101112713). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them. The work of Zhang was partially supported by the NSFC (grant number 62161146001).

References

1. Amir, G., et al.: Verifying Learning-Based Robotic Navigation Systems: Supplementary Video (2022). https://youtu.be/QIZqOgxLkAE
2. Amir, G., et al.: Verifying learning-based robotic navigation systems. In: Proceedings of the 29th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), pp. 607–627 (2023)

3. Amir, G., Schapira, M., Katz, G.: Towards scalable verification of deep reinforcement learning. In: Proceedings of the 21st International Conference on Formal Methods in Computer-Aided Design (FMCAD), pp. 193–203 (2021)
4. Bak, S., Tran, H.D., Hobbs, K., Johnson, T.T.: Improved geometric path enumeration for verifying ReLU neural networks. In: International Conference on Computer Aided Verification, pp. 66–96. Springer (2020)
5. Balunovic, M., Baader, M., Singh, G., Gehr, T., Vechev, M.: Certifying geometric robustness of neural networks. Adv. Neural Inf. Process. Syst. **32** (2019)
6. Barrett, C., et al.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 171–177. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_14
7. Bassan, S., Amir, G., Corsi, D., Refaeli, I., Katz, G.: Formally explaining neural networks within reactive systems. In: Proceedings of the 23rd International Conference on Formal Methods in Computer-Aided Design (FMCAD), pp. 10–22 (2023)
8. Bassan, S., Katz, G.: Towards formal XAI: formally approximate minimal explanations of neural networks. In: Proceedings of the 29th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), pp. 187–207 (2023)
9. Bauer-Marquart, F., Boetius, D., Leue, S., Schilling, C.: SpecRepair: counter-example guided safety repair of deep neural networks. In: Legunsen, O., Rosu, G. (eds.) Model checking software: 28th International Symposium, SPIN 2022, Virtual Event, May 21, 2022, Proceedings, pp. 79–96. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-15077-7_5
10. Boyd, S.P., Vandenberghe, L.: Convex Optimization. Cambridge University Press (2004)
11. Brix, C., Bak, S., Liu, C., Johnson, T.T.: The fourth international verification of neural networks competition (VNN-COMP 2023): summary and results. arXiv preprint arXiv:2312.16760 (2023)
12. Christakis, M., et al.: Automated safety verification of programs invoking neural networks. In: International Conference on Computer Aided Verification, pp. 201–224. Springer (2021)
13. Cimatti, A., Griggio, A., Irfan, A., Roveri, M., Sebastiani, R.: Incremental linearization for satisfiability and verification modulo nnlinear arithmetic and transcendental functions. ACM Trans. Computat. Logic **19**(3), 1–52 (2018)
14. Desmartin, R., Isac, O., Passmore, G., Stark, K., Komendantskaya, E., Katz, G.: Towards a certified proof checker for deep neural network verification. In: Proceedings of the 33rd International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR), pp. 198–209 (2023)
15. Eliyahu, T., Kazak, Y., Katz, G., Schapira, M.: Verifying learning-augmented systems. In: Proceedings of the Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM), pp. 305–318 (2021)
16. Ferrari, C., Mueller, M.N., Jovanović, N., Vechev, M.: Complete verification via multi-neuron relaxation guided branch-and-bound. In: International Conference on Learning Representations (2022)
17. Funk, N., Baumann, D., Berenz, V., Trimpe, S.: Learning event-triggered control from data through joint optimization. IFAC J. Syst. Control **16** (2021)
18. Geng, C., Le, N., Xu, X., Wang, Z., Gurfinkel, A., Si, X.: Towards reliable neural specifications. In: International Conference on Machine Learning, pp. 11196–11212. PMLR (2023)

19. Gopinath, D., Converse, H., Pasareanu, C., Taly, A.: Property inference for deep neural networks. In: 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 797–809. IEEE (2019)
20. Gowal, S., et al.: On the effectiveness of interval bound popagation for training verifiably robust models. arXiv preprint arXiv:1810.12715 (2018)
21. Graph Neural Networks support in ONNX (2022). https://github.com/microsoft/onnxruntime/issues/12103
22. Guidotti, D., Leofante, F., Pulina, L., Tacchella, A.: Verification of neural nNetworks: enhancing scalability through pruning. In: European Conference on Artificial Intelligence, pp. 2505–2512. IOS Press (2020)
23. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2023). https://www.gurobi.com
24. Henriksen, P., Lomuscio, A.: DEEPSPLIT: an eEfficient splitting method for neural network verification via indirect effect analysis. In: International Joint Conference on Artificial Intelligence, pp. 2549–2555. ijcai.org (2021)
25. Henriksen, P., Lomuscio, A.R.: Efficient neural network verification via adaptive refinement and adversarial search. In: Giacomo, G.D., et al. (eds.) European Conference on Artificial Intelligence, vol. 325, pp. 2513–2520. IOS Press (2020)
26. Huang, X., Marques-Silva, J.: From robustness to explainability and back again. arXiv preprint arXiv:2306.03048 (2023)
27. Huang, X., Liu, M.-Y., Belongie, S., Kautz, J.: Multimodal unsupervised image-to-image translation. In: Ferrari, V., Hebert, M., Sminchisescu, C., Weiss, Y. (eds.) ECCV 2018. LNCS, vol. 11207, pp. 179–196. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01219-9_11
28. Ignatiev, A., Narodytska, N., Marques-Silva, J.: Abduction-based explanations for machine learning models. In: AAAI Conference on Artificial Intelligence, vol. 33, pp. 1511–1519. AAAI Press (2019)
29. Isac, O., Barrett, C., Zhang, M., Katz, G.: Neural network verification with proof production. In: Proceedings of the 22nd International Conference on Formal Methods in Computer-Aided Design (FMCAD), pp. 38–48 (2022)
30. Julian, K., Kochenderfer, M., Owen, M.: Deep neural network compression for aircraft collision avoidance systems. J. Guid. Control. Dyn. **42**(3), 598–608 (2019)
31. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: an efficient SMT solver for verifying deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 97–117. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_5
32. Katz, G., et al.: The Marabou framework for verification and analysis of deep neural networks. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11561, pp. 443–452. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_26
33. Katz, S.M., Corso, A.L., Strong, C.A., Kochenderfer, M.J.: Verification of image-based neural network controllers using generative models. J. Aerosp. Inf. Syst. **19**(9), 574–584 (2022)
34. Liu, C., Cofer, D., Osipychev, D. Verifying an aircraft collision avoidance neural network with Marabou. In: Rozier, K.Y., Chaudhuri, S. (eds.) NFM 2023. LNCS, pp. 79–85. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-33170-1_5
35. Lopez, D.M., Choi, S.W., Tran, H.-D., Johnson, T.T.: NNV 2.0: the neural network verification tool. In: Enea, C., Lal, A. (eds.) CAV 2023, pp. 397–412. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-37703-7_19
36. Lundberg, S.M., Lee, S.I.: A unified approach to interpreting model predictions. Adv. Neural Inf. Process. Syst. **30** (2017)

37. Malfa, E.L., Michelmore, R., Zbrzezny, A.M., Paoletti, N., Kwiatkowska, M.: On guaranteed optimal robust explanations for NLP models. In: International Joint Conference on Artificial Intelligence, pp. 2658–2665. ijcai.org (2021)
38. Matheson, R.: AI system optimally allocates workloads across thousands of servers to cut costs, save energy. Tech Xplore (2019). https://techxplore.com/news/2019-08-ai-optimally-allocates-workloads-thousands.html
39. Mirman, M., Hägele, A., Bielik, P., Gehr, T., Vechev, M.: Robustness certification with generative models. In: ACM SIGPLAN International Conference on Programming Language Design and Implementation, pp. 1141–1154 (2021)
40. Mohapatra, J., Weng, T.W., Chen, P.Y., Liu, S., Daniel, L.: Towards verifying robustness of neural networks against a family of semantic perturbations. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 244–252 (2020)
41. Müller, M.N., Makarchuk, G., Singh, G., Püschel, M., Vechev, M.: Prima: general and precise neural network certification via scalable convex hull approximations. Proc. ACM Program. Lang. **6**(POPL), 1–33 (2022)
42. Paterson, C., et al.: DeepCert: verification of contextually relevant robustness for neural network image classifiers. In: Habli, I., Sujan, M., Bitsch, F. (eds.) SAFECOMP 2021. LNCS, pp. 3–17. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-83903-1_5
43. Ribeiro, M.T., Singh, S., Guestrin, C.: Why should i trust you? Explaining the predictions of any classifier. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1135–1144 (2016)
44. Ribeiro, M.T., Singh, S., Guestrin, C.: Anchors: high-precision model-agnostic explanations. In: AAAI Conference on Artificial Intelligence, vol. 32, pp. 1527–1535. AAAI Press (2018)
45. Singh, G., Ganvir, R., Püschel, M., Vechev, M.: Beyond the single neuron convex barrier for neural network certification. Adv. Neural. Inf. Process. Syst. **32**, 15098–15109 (2019)
46. Singh, G., Gehr, T., Püschel, M., Vechev, M.: An abstract domain for certifying neural networks. Proc. ACM Program. Lang. **3**(POPL), 1–30 (2019)
47. Singh, G., Gehr, T., Püschel, M., Vechev, M.: Boosting robustness certification of neural networks. In: International Conference on Learning Representations (2019)
48. Strong, C., et al.: Global optimization of objective functions represented by ReLU networks. J. Mach. Learn. **112**(10), 3685–3712 (2021)
49. Sun, Y., Usman, M., Gopinath, D., Păsăreanu, C.S.: VPN: verification of poisoning in neural networks. In: Isac, O., Ivanov, R., Katz, G., Narodytska, N., Nenzi, L. (eds.) Software Verification and Formal Methods for ML-Enabled Autonomous Systems: 5th International Workshop, FoMLAS 2022, and 15th International Workshop, NSV 2022, Haifa, 31 July–1 August, and 11 August 2022, Proceedings, pp. 3–14. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-21222-2_1
50. Tjeng, V., Xiao, K.Y., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: International Conference on Learning Representations (2019)
51. Tran, H.D., et al.: NNV: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In: International Conference on Computer Aided Verification, pp. 3–17. Springer (2020)
52. Vanderbei, R.: Linear programming: foundations and extensions. J. Oper. Res. Soc. (1998)
53. Vaswani, A., et al.: Attention is all nou need. Adv. Neural Inf. Process. Syst. **30** (2017)

54. Vinzent, M., Sharma, S., Hoffmann, J.: Neural policy safety verification via predicate abstraction: CEGAR. In: AAAI Conference on Artificial Intelligence, pp. 15188–15196. AAAI Press (2023)
55. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Efficient formal safety analysis of neural networks. Adv. Neural. Inf. Process. Syst. **31**, 6369–6379 (2018)
56. Wang, S., et al.: Beta-crown: efficient bound propagation with per-neuron split constraints for neural network robustness verification. Adv. Neural. Inf. Process. Syst. **34**, 29909–29921 (2021)
57. Wu, H., et al.: Parallelization techniques for verifying neural networks. In: Proceedings of the 20th International Conference on Formal Methods in Computer-Aided Design (FMCAD), pp. 128–137 (2020)
58. Wu, H., Zeljić, A., Katz, G., Barrett, C.: Efficient neural network analysis with sum-of-infeasibilities. In: Proceedings of the 28th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), pp. 143–163 (2022)
59. Wu, H., Barrett, C., Sharif, M., Narodytska, N., Singh, G.: Scalable verification of GNN-based job schedulers. Proc. ACM Program. Lang. **6**(OOPSLA), 1036–1065 (2022)
60. Wu, H., et al.: Artifact for Marabou 2.0: a versatile formal analyzer of neural networks (2022). https://doi.org/10.5281/zenodo.11116016
61. Wu, H., et al.: Marabou 2.0: a versatile formal analyzer of neural networks. arXiv preprint arXiv:2401.14461 (2024)
62. Wu, H., et al.: Toward certified robustness against real-world distribution shifts. In: IEEE Conference on Secure and Trustworthy Machine Learning, pp. 537–553. IEEE (2023)
63. Wu, M., Wu, H., Barrett, C.: VeriX: towards verified explainability of deep neural networks. Adv. Neural Inf. Process. Syst. (2022)
64. Xie, X., Kersting, K., Neider, D.: Neuro-symbolic verification of deep neural networks. In: International Joint Conferences on Artificial Intelligence, pp. 3622–3628. ijcai.org (2022)
65. Xu, K., et al.: Automatic perturbation analysis for scalable certified robustness and beyond. Adv. Neural. Inf. Process. Syst. **33**, 1129–1141 (2020)
66. Yerushalmi, R.: Enhancing deep reinforcement learning with executable specifications. In: International Conference on Software Engineering, pp. 213–217. IEEE (2023)
67. Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., Lipson, H.: Understanding neural networks through deep visualization. arXiv preprint arXiv:1506.06579 (2015)
68. Zelazny, T., Wu, H., Barrett, C., Katz, G.: On reducing over-approximation errors for neural network verification. In: Proceedings of the 22nd International Conference on Formal Methods in Computer-Aided Design (FMCAD), pp. 17–26 (2022)
69. Zhang, H., et al.: General cutting planes for bound-propagation-based neural network verification. Adv. Neural. Inf. Process. Syst. **35**, 1656–1670 (2022)
70. Zhang, H., Weng, T.W., Chen, P.Y., Hsieh, C.J., Daniel, L.: Efficient neural network robustness certification with general activation functions. Adv. Neural. Inf. Process. Syst. **31**, 4944–4953 (2018)