CAISAR: A platform for Characterizing Artificial Intelligence Safety and Robustness

Julien Girard-Satabin, Michele Alberti, François Bobot, Zakaria Chihani, Augustin Lemesle

CAISAR: A platform for Characterizing Artificial Intelligence Safety and Robustness

Michele Alberti (michele.alberti@cea.fr)[†] , **François Bobot (francois.bobot@cea.fr)**[†] , **Zakaria Chihani** , **Julien Girard-Satabin (julien.girard2@cea.fr)**[†] , **Augustin Lemesle**

Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France

†: corresponding author

Abstract

We present CAISAR, an open-source platform under active development for the characterization of AI systems' robustness and safety. CAISAR provides a unified entry point for defining verification problems by using WhyML, the mature and expressive language of the Why3 verification platform. Moreover, CAISAR orchestrates and composes state-of-the-art machine learning verification tools which, individually, are not able to efficiently handle all problems but, collectively, can cover a growing number of properties. Our aim is to assist, on the one hand, the V&V process by reducing the burden of choosing the methodology tailored to a given verification problem, and on the other hand the tools developers by factorizing useful features – visualization, report generation, property description – in one platform. CAISAR will soon be available at https://git.frama-c.com/pub/caisar.

1 Introduction

The integration of machine learning programs as components of critical systems is said to be bound to happen; initiatives from various private and govermental actors (*e.g.,* US' NSF funding for Trustworthy AI[1], France's Grand Défi IA de confiance[2]) are a consequence of that fact. Trusting such programs is thus becoming a crucial issue, both on technical and ethical sides.

A possible approach to trust is formal test and verification, a broad set of techniques and tools that have been applied to software safety for several decades. These formal methods build on sound mathematical foundations to assess the behaviour of programs in a principled way, be it for generating tests or providing proven guarantees. For the last few years, several independent works have started to investigate the possible applications of formal verification to machine learning program verification and its limitations. This led to what could be characterized as a Cambrian explosion of tools aiming to solve a particular subset of the machine learning verification field. In less than five years, more than 20 tools were produced, upgraded or abandoned. These tools use different techniques, different input formats, handle different ML artefacts and, most importantly, have varying performances depending on the problems. Our goal of orchestrating multiple tools aims at maximizing the property coverage in the context of a validation and verification process.

To this end, we present a platform dedicated to Characterizing Artificial Intelligence Safety and Robustness (CAISAR) that aims to unify several formal methods and tools, at the input, through the use of a mature and expressive property description and proof orchestration language, at the output, through the factorization of features such as visualization and report generation, and at the usage, through shared heuristics and interconnections between tools.

The answer to the question "To what extent can I trust my machine learning program?" has many components, ranging from data analysis to decision explainability. One such important components is dealing with verification and validation, and we wish to make CAISAR an important element in the safety toolbelt by covering these applications.

In the following, we will first present the design principles of CAISAR and state its main goals. We will follow by a description of its most prominent features, as well as its limitations. We will then explain the position of CAISAR regarding other tools for formal verification of machine learning programs, and conclude by presenting some future work and possible research problems.

2 Core principles of CAISAR

The aim for CAISAR is to provide a verification environment for Artificial Intelligence (AI) based systems tailored to different needs. The profusion of tools for AI-programs certification offers numerous possibilities, from the choice of technology (formal methods, test generation) to the scope of properties to check (coverage, suitability to a given distribution, robustness). However, with increased possibilites comes the burden of choice. Which method better suits a given use case? Are the results provided by this particular method trustworthy enough? How to bring trust in the process of selecting, tailoring and computing results of a given tool? How to evaluate a given tool against others? Those are the questions that we aim to answer with CAISAR.

[1] https://www.nsf.gov/pubs/2022/nsf22502/nsf22502.htm

[2] https://www.gouvernement.fr/grand-defi-securiser-certifier-et-fiabiliser-les-systemes-fondes-sur-l-intelligence-artificielle

2.1 Compatibility with existing and future methods

The first principle of CAISAR is to ease this burden of choice by automating parts of it. CAISAR aims to provide a unique interface for vastly different tools, with a single entry point to specify verification goals. Choosing which tool to use is an informed decision that may not be relevant for the user; the goal is to provide an actionable answer on the safety of the system, by using whatever tool is suitable for the problem. Ideally, the user should not be bothered with deciding which tool is suitable for their use case: CAISAR will automatically figure out how to express the given property to suit verifiers. As AI systems pipelines are becoming more and more complex, it is crucial for CAISAR to handle this complexity. Currently, CAISAR supports neural networks and support vector machines, and an industrial benchmark of an ensemble model (NN-SVM), which we are unable to further discuss, is being used as a concrete real-word use-case.

2.2 Common modelling and answers

Existing verifiers rely on different decision procedures, *e.g.,* Mixed Integer Linear Programming (MILP), abstract interpretation, or Satisfaction Modulo Theory (SMT) calculus. Modelling a verification problem using these frameworks require different skills and is time-consuming; for instance, some modelling choices made for MILP may not be applied under SMT. Moreover, even if one succeeds in phrasing a verification problem under multiple decision procedures, the different results may not be immediately comparable.

CAISAR aims to provide a common ground for inputs and outputs, which will lead to an easier comparison, lower time consumption and an informed decision. Furthermore, collecting and presenting the user with multiple answers from different techniques can provide additional confidence on the studied system. In order for users to trust CAISAR as well, it needs to rely on well-known and approved principles and technologies. It is developed in OCaml, a strongly typed programming language, used to develop tools for program verification and validation. Such tools include CompCert[Leroy *et al.*, 2016], a C compiler that is guaranteed to output C-ANSI compliant source code, Frama-C [Baudin *et al.*, 2021], a platform for the static analysis of C code, and Why3 [Filliâtre and Paskevich, 2013], a platform for deductive verification of programs.

2.3 Tools composition

Some works are starting to combine multiple techniques [Singh *et al.*, 2019] for their analysis, using an exact MILP solver to refine bounds obtained by abstract interpretation. Our goal with CAISAR is to bring tool composition to another level. For instance, metamorphic transformations could generate different input space partitions for formal verifiers. A reachability analysis tool could be called numerous times with tighter bounds until reaching a precise enough answer. Coverage testing objectives could be extracted from reachability analysis tools and fed to test generators. CAISAR will be more than the sum of its part, allowing communication between vastly different tools to provide faster and more accurate answers.

2.4 Automatic proposal of verification strategies

A long-term goal for CAISAR is to provide a reasoning engine where past verification problems processed by CAISAR can inform next ones, gradually building a knowledge base that is suitable for the specific needs of the user. CAISAR will also implement its own built-in heuristics to supplement specialized programs that do not implement them.

3 Architecture and features

CAISAR's architecture can be divided into the following functional blocks:

1. A Common specification Language (CL)
2. A Proof Obligation Generator (POG), associated with a Dispatcher (DISP)
3. An Intermediate Representation (IR)
4. A visualization module (VIZ)

See fig. 1 for a visual depiction of dependencies between blocks.

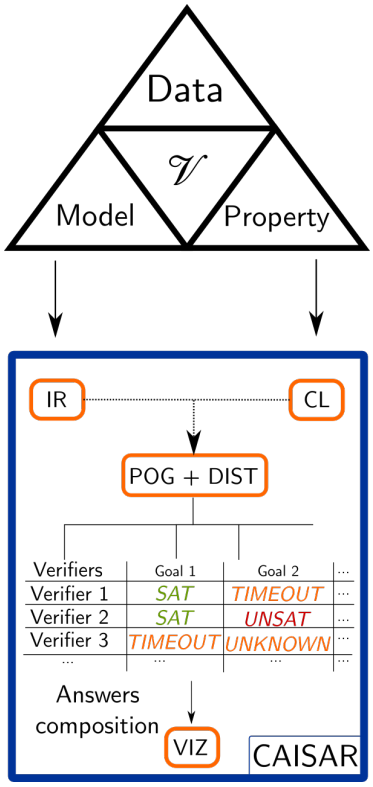

Figure 1: CAISAR overall architecture

3.1 Specification language and verification predicates

A typical task for program verification involves to solve a *verification problem*. A verification problem consists on checking that a program with a given set of inputs is meeting certain expectations on its outputs. More formally, let $\mathcal{X}$ be an input

space, $\mathcal{P}(\mathcal{X})$ be a property on the input space, f be a program, $\mathcal{Y}$ be the image of $\mathcal{X}$ by f and $\mathcal{Q}(\mathcal{Y})$ be a property on the output space. By property, we mean a statement that describes a desirable behaviour for the program. Let $\mathcal{V} = (\mathcal{X}, f, \mathcal{P}, \mathcal{Q})$ be a *verification problem*. The goal is to verify the following property:

$$\forall x \in \mathcal{X}, \mathcal{P}(x) \Rightarrow \mathcal{Q}(f(x))$$

To write $\mathcal{V}$, one needs to be able to express concisely and without ambiguity each component: the program to verify f, the properties $\mathcal{P}$ and $\mathcal{Q}$, and the dataset $\mathcal{X}$. To this end, CAISAR provides full support for the WhyML specification and programming language [Filliâtre and Paskevich, 2013]. WhyML is a language with static strong typing, pattern matching, types invariants and inductive predicates. This gives WhyML programs a sound semantic as logical propositions. WhyML is at the core of the Why3 verification platform, and has been used as an intermediate language for verification of Ada programs [Guitton *et al.*, 2011]. This global expressiveness and safety allows to write $\mathcal{V}$ once and for all, independently of the verifier. For instance, Figure 2 shows the definition of robustness against a perturbation of amplitude ε using the l_∞ distance within CAISAR's standard library, and fig. 3 the WhyML file the user need to write in order to verify the robustness of a given TestSVM against a perturbation of amplitude 0.5. Note that all the necessary element to define $\mathcal{V}$, namely f, $\mathcal{X}$, $\mathcal{P}$ and $\mathcal{Q}$, are defined in those files: f is the function TestSVM, $\mathcal{Y}$ is the image of svm_apply (a function that describes the application of a SVM on a particular input a), and $\mathcal{Q}$ is the predicate itself. Note that WhyML is not limited to the robustness against a given perturbation property, often met in the literature. For instance, asserting that a neural network respect the properties of being differentialy private [Abadi *et al.*, 2016] or respecting causal fairness [Urban *et al.*, 2019] is something that could be phrased as WhyML programs, since those properties have a mathematical characterization. Finally, WhyML does not constrain the form of $\mathcal{P}$ nor $\mathcal{Q}$. In particular, it is possible to define multiple verification goals in the same $\mathcal{V}$, opening the way to subdivide it into subproblems, and providing answers at each step.

CAISAR then automatically translates $\mathcal{V}$ into a format supported by the selected verifiers, through a succession of encoding transformations and simplifications. For instance, some verifiers are best used when trying to falsify the property: instead of checking

$$\forall x \in \mathcal{X}, \mathcal{P}(x) \Rightarrow \mathcal{Q}(f(x))$$

checking the negation

$$\exists x \in \mathcal{X}, \mathcal{P}(x) \nRightarrow \mathcal{Q}(f(x))$$

This transformation is embedded in CAISAR, when calling Marabou: a verification problem $\mathcal{V}$ can be transformed into an equivalent one $\mathcal{V}'$ that can be dispatched to Marabou.

3.2 Proof Obligations Generations for various tools

CAISAR currently supports a variety of tools and techniques: metamorphic testing, reachability analysis based on abstract interpretation and constraint-based propagations. CAISAR can analyze neural networks and support vector machines. This versatility allows for CAISAR to verify system components using different machine learning architectures.

Marabou

Marabou [Katz *et al.*, 2019] is a deep neural network verification complete verifier. Its core routine relies on a modified simplex algorithm that lazily relaxes constraints on piecewise linear activation functions. Marabou also makes use of several heuristics that help speeding up the verification procedure, like relying on tight convex overapproximations [Wu *et al.*, 2022] or sound overapproximations [Ostrovsky *et al.*, 2022]. It can answer reachability and conjunction of linear constraint queries. Marabou ranked fifth at the VNN-COMP 2021. It is currently in active development.

SAVer

The Support Vector Machine reachability analysis tool SAVer [Ranzato and Zanella, 2019] is specialized in the verification of support vector machines (SVM), a popular machine learning algorithm used alongside neural networks for classification or regression tasks. SAVer can answer reachability queries, and supports a variety of SVM configurations. This tool was selected for support as, to the best of our knowledge, it is the first one to deal with verification of SVM.

Alt-Ergo and SMTLIB compliant solvers

Existing general purpose SMT solvers for program verification like Alt-Ergo [Conchon *et al.*, 2018] or Z3 [de Moura

```
type input_type = int -> t
type output_type = int
type model = {
          app :
          input_type -> output_type;
          num_input: int;
          num_classes: int
}

predicate dist_linf
  (a: input_type)
  (b: input_type)
  (eps:t)
  (n: int) =
  forall i. 0 <= i < n ->
    .- eps .< a i .- b i .< eps

predicate robust_to
  (model: model)
  (a: input_type)
  (eps: t) =
  forall b. dist_linf a b eps
  model.num_input ->
  model.app a = model.app b
```

Figure 2: An example of a predicate in CAISAR's standard library: being "robust to" against a perturbation of amplitude ε. Here, the predicate defines $\mathcal{Q}(\mathcal{Y})$.

```
use TestSVM.SVMasArray
use ieee_float.Float64
use caisar.SVM

goal G: forall a : input_type.
robust_to svm_apply a (0.5:t)
```

Figure 3: Example verification problem specified to CAISAR. The program to verify is TestSVM, the input space is defined by the elements in a, the output space is the result of the application of the function svm_apply.

and Bjørner, 2008] all support a standard input language, SMTLIB [Barrett *et al.*, 2016]. CAISAR leverages Why3 existing support for SMT solvers and can translate neural network control flows directly into SMTLIB compliant strings using its intermediate representation, which allows the support of a variety of off-the-shelf solvers. Note that the VNNLIB standard, used in the VNN-COMP, uses a subset of SMTLIB2, which paves the way for the support of future tools in CAISAR.

Python Reachability Assessment Tool

The Python Reachability Assessment Tool (PyRAT) is a static analyzer targeting specifically neural networks. It builds upon the framework of abstract interpretation [Cousot and Cousot, 1977] using abstract domains adapted for the approximation of the reachable space in a neural network. Three main domains are used: intervals with symbolic relations as described in [Li *et al.*, 2019; Wang *et al.*, 2018], zonotopes [Singh *et al.*, 2018] and Deep poly domain [Singh *et al.*, 2019].

For low dimensional inputs, PyRAT use input partitioning as described in [Wang *et al.*, 2018], with heuristics tailored to relational domains: the zonotope domain and the deeppoly domain with backsubstitution. Those heuristics allow the computation of a non-trivial (*e.g.,* not just widest interval first) score ranking the inputs by their estimated influence on the outputs. PyRAT has comparable results to state-of-the-art analyzers on the widely used ACAS-Xu [Manfredi and Jestin, 2016] benchmark, and outperforms the similar domains of ERAN on S-shape activations functions such as the sigmoid or hyperbolic tangent functions with specific approximations.

AIMOS: a Metamorphic testing utility

AI Metamorphism Observing Software (AIMOS) is a software developped at the same time as CAISAR, aiming to provide metamorphic properties testing or perturbations on a dataset for a given AI model. Metamorphic testing is a testing technique relying on properties symmetries and invariance on the operating domain. See [Chen *et al.*, 2018] for a comprehensive survey on this approach. AIMOS offers tools to derive properties from a set of transformations on the inputs: given $\mathcal{P}$, $\mathcal{Q}$, $\mathcal{X}$ and a transformation function $t_\theta : x \in \mathcal{X} \mapsto \mathcal{X}$, it generates a set of new properies $\mathcal{Q}^{'}$ that are coherent with the transformation. As an example, a symmetry on the inputs of a classification model could result on a symmetry on the outputs; AIMOS would then automatically modify the property to check against the symmetrical labels.

AIMOS can generate test cases scenarios from the most common input transformations (geometrical rotations, noise addition); others can be added if necessary. AIMOS was evaluated on a metamorphic property on the ACAS-Xu benchmark. The aim of the property was to evaluate the ability of neural networks trained on ACAS to generalize with symmetric inputs. Given a symmetry on inputs, AIMOS generates the expected symmetrical output, and tests models against the base and symmetrical outputs. See table 1 for results. AIMOS was able to show that neural networks trained on ACAS have a low, but noticeable sensitivity to symmetry on one input.

Model previous answer	Percentage of identical answer
COC	89.7%
WL	95.9%
WR	99.6%
L	95.3%
R	99.8%

Table 1: Average number of same answer for all 45 models of the ACAS-Xu benchmark, computed by AIMOS. First column denotes values presented in the benchmark.

3.3 Supported formats

CAISAR supports all input formats used by its integrated verifiers. Most verifiers require either a framework-specific binary (Pytorch's pth, Tensorflow tf), a custom description language (NNet), or an Open Neural Network eXchange (ONNX) [3] file. CAISAR is able to parse any of these input formats and extract useful metadata for the building of the verification strategy. It can also output a verification problem into the SMTLIB [Barrett *et al.*, 2016] format, supported by all general purpose solvers, as well as in the ONNX format. The VNN-Lib initiative [4] provides a standard format for verification problems that relies on SMTLIB; thus CAISAR also supports VNN-Lib. CAISAR aims for maximum interoperability, and can be used as a hub to write and convert verification queries adapted to different verifiers. Additionally, verifiers sometimes require datasets to verify properties against, especially reachability analysis tools. As such, CAISAR currently supports datasets as flattened features under a csv file, and RGB images.

3.4 Answer composition

CAISAR currently offer two ways to compose verifiers. First, CAISAR can launch several solvers on the same task and compose their answer: it can then provide a summary stating which solver succeeded and which one failed. Second, CAISAR has the ability to verify pipelines that are composed of several machine learning programs: for instance, a pipeline composed of several neural networks, or a neural network which outputs are processed by a SVM. CAISAR can be used to state an overall verification goal, and to model that the outputs of a block of the pipeline are the inputs of another block.

[3] https://onnx.ai/

[4] http://www.vnnlib.org/

More advanced methods of composition, such as automatic subgoals generation or refinement by multiple analysis constitute a promising research venue.

4 Background & related works

In less than five years, a profusion of tools and techniques leveraging formal verification to provide trust on neural network sprouted [Katz *et al.*, 2017; Katz *et al.*, 2019; Wang *et al.*, 2018; Wang *et al.*, ; Singh *et al.*, 2018; Singh *et al.*, 2019; Shi *et al.*, 2020; Bak, 2021; Henriksen and Lomuscio, 2020; Dutta *et al.*, 2017; Palma *et al.*, ; Urban *et al.*, 2019; Ehlers, 2017]. See [Urban and Miné, 2021; Liu *et al.*, 2019] for more comprehensive surveys on the verification and validation of machine learning programs.

As for general purpose verification platforms, examples include the Why3 deductive verification platform and the Frama-C [Baudin *et al.*, 2021] C static analysis platform. We leverage multiple existing features of Why3, such as the WhyML language support, transformation and rewriting engine. Why3 and Frama-C both lack the interfaces and tooling to handle neural network. Experiments we conducted involving the EVA Frama-C plugin applied on simple reachability analysis properties showed a lack of scalability that a naive python reachability analysis tool, specialized in neural networks, was able to overcome quickly. Conversion of neural networks in C programs that were scalable for EVA presented difficult challenges. The differing structure between C programs and neural networks implies differing verification problems. Thus, it seems more fruitful to investigate a specialized platform for machine learning programs.

The ProDeep platform [Li *et al.*, 2020] aims to regroup several verifiers under a single user interface. It provides a single entry point, supports input formats and offers numerous visualization tools. It does not aim to provide other properties than those that are natively supported by its embedded verifiers. It also supports a fixed set of datasets. They make use of DeepG [Balunovic *et al.*, 2019] to generate constraints for verifiers, effectively combining tools. Their scope seems limited to neural networks, whereas CAISAR currently supports neural networks and support vector machines, and aims to support a wider set of machine learning models.

The most similar work to CAISAR is the DNNV platform [Shriver *et al.*, 2021]. As CAISAR, DNNV provides support to various state-of-the-art verifiers. It similarly aims to be a hub for neural network verification by supporting a wide range of input and output formats, and by providing a modelling language for properties specification and discharge to capable provers. Their Domain Specific Language, DNNP, is built on Python; while CAISAR's specification language, WhyML, is already used in several formal verification platforms and provide additional theoretical guarantees, which is a key component to provide trust. As stated before, WhyML allows specifying multiple verification goals in the same verification problem, which helps modelling more complex use cases.

The main difference between CAISAR and DNNV is that the latter does not combine verifiers answers, that is to say there is (at the time of writing) no feature that aims to interoperate verifiers: from the DNNV documentation[5]: "DNNV standardizes the network and property input formats to enable multiple verification tools to run on a single network and property. This facilitates both verifier comparison, and artifact re-use." As verifiers are becoming more and more sophisticated and specialized, combination of methods will become even more fruitful, and we expect this to be a key difference with DNNV.

5 Conclusion & future works

As the field of machine learning verification is blooming, choosing the right tool for the right verification problem becomes more and more tedious. We presented CAISAR, a platform aimed to alleviate this difficulty by presenting a single, extensible entry point to machine learning verification problem modelling and solving. Plenty of work still needs to be done, however.

Altough CAISAR already integrates some state-of-the-art tools, other verifiers that ranked high in the VNN-COMP are on the way of integration. Such verifiers include α, β-CROWN [Wang *et al.*, ; Xu *et al.*, 2021], who scored first on said competition.

Another research venue would be the integration of neural network reparation techniques such as [Goldberger *et al.*, 2020]. Corrective techniques would contribute to provide a feedback loop composed of problem specification, verification, fault identification and correction proposal.

Various problem splitting heuristics based, for instance, on [Girard-Satabin *et al.*, 2021; Bunel *et al.*, 2020] could be integrated into CAISAR to leverage parallelism for verifiers that do not support them

Data is the cornerstone of modern machine learning systems, and it is necessary to give tools to handle its complexity. Support for more various data kinds, such as time series, is a first step towards this direction. Integration of tools for analyzing data *in relation with* a program, for instance out-of-distribution detection, is another future work.

Finally, to further help the user to select the optimal set of tools for its verification problem, a long-term goal of CAISAR is to provide a verification helper to design optimal queries for verification problems based on previous runs.

References

[Abadi *et al.*, 2016] Martin Abadi, Andy Chu, Ian Goodfellow, Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep Learning with Differential Privacy. In *23rd ACM Conference on Computer and Communications Security (ACM CCS)*, pages 308–318, 2016.

[Bak, 2021] Stanley Bak. Nnenum: Verification of ReLU Neural Networks with Optimized Abstraction Refinement. In Aaron Dutle, Mariano M. Moscato, Laura Titolo, César A. Muñoz, and Ivan Perez, editors, *NASA Formal Methods*, Lecture Notes in Computer Science, pages 19–36, Cham, 2021. Springer International Publishing.

[Balunovic *et al.*, 2019] Mislav Balunovic, Maximilian Baader, Gagandeep Singh, Timon Gehr, and Martin

[5] https://docs.dnnv.org/en/stable/

Vechev. Certifying Geometric Robustness of Neural Networks. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[Barrett *et al.*, 2016] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2016.

[Baudin *et al.*, 2021] Patrick Baudin, François Bobot, David Bühler, Loïc Correnson, Florent Kirchner, Nikolai Kosmatov, André Maroneze, Valentin Perrelle, Virgile Prevosto, Julien Signoles, and Nicky Williams. The dogged pursuit of bug-free C programs: The Frama-C software analysis platform. *Communications of the ACM*, 64(8):56–68, August 2021.

[Bunel *et al.*, 2020] Rudy Bunel, Jingyue Lu, Ilker Turkaslan, Philip H.S. Torr, Pushmeet Kohli, and M. Pawan Kumar. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21(42):1–39, 2020.

[Chen *et al.*, 2018] Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, Pak-Lok Poon, Dave Towey, TH Tse, and Zhi Quan Zhou. Metamorphic testing: A review of challenges and opportunities. *ACM Computing Surveys (CSUR)*, 51(1):1–27, 2018.

[Conchon *et al.*, 2018] Sylvain Conchon, Albin Coquereau, Mohamed Iguernlala, and Alain Mebsout. Alt-Ergo 2.2. In *SMT Workshop: International Workshop on Satisfiability Modulo Theories*, Oxford, United Kingdom, July 2018.

[Cousot and Cousot, 1977] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252, 1977.

[de Moura and Bjørner, 2008] Leonardo de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, pages 337–340, Berlin, Heidelberg, 2008. Springer.

[Dutta *et al.*, 2017] Souradeep Dutta, Susmit Jha, Sriram Sanakaranarayanan, and Ashish Tiwari. Output Range Analysis for Deep Neural Networks. *arXiv:1709.09130 [cs, stat]*, September 2017.

[Ehlers, 2017] Ruediger Ehlers. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. *arXiv:1705.01320 [cs]*, May 2017.

[Filliâtre and Paskevich, 2013] Jean-Christophe Filliâtre and Andrei Paskevich. Why3 - Where Programs Meet Provers. In Matthias Felleisen and Philippa Gardner, editors, *Programming Languages and Systems*, Lecture Notes in Computer Science, pages 125–128, Berlin, Heidelberg, 2013. Springer.

[Girard-Satabin *et al.*, 2021] Julien Girard-Satabin, Aymeric Varasse, Marc Schoenauer, Guillaume Charpiat, and Zakaria Chihani. DISCO: Division of input space into convex polytopes for neural network verification. *JFLA*, 2021.

[Goldberger *et al.*, 2020] Ben Goldberger, Guy Katz, Yossi Adi, and Joseph Keshet. Minimal modifications of deep neural networks using verification. In Elvira Albert and Laura Kovacs, editors, *LPAR23. LPAR-23: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 73 of *EPiC Series in Computing*, pages 260–278. EasyChair, 2020.

[Guitton *et al.*, 2011] Jérôme Guitton, Johannes Kanig, and Yannick Moy. why hi-lite ada. *Rustan, et al.[32]*, pages 27–39, 2011.

[Henriksen and Lomuscio, 2020] P Henriksen and A Lomuscio. Efficient Neural Network Verification via Adaptive Refinement and Adversarial Search. In *24th European Conference on Artificial Intelligence - ECAI 2020*, page 8, Santiago de Compostela, Spain, 2020.

[Katz *et al.*, 2017] Guy Katz, Clark Barrett, David Dill, Kyle Julian, and Mykel Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. *arXiv preprint arXiv:1702.01135*, 2017.

[Katz *et al.*, 2019] Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, David L. Dill, Mykel J. Kochenderfer, and Clark Barrett. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification*, volume 11561, pages 443–452. Springer International Publishing, Cham, 2019.

[Leroy *et al.*, 2016] Xavier Leroy, Sandrine Blazy, Daniel Kästner, Bernhard Schommer, Markus Pister, and Christian Ferdinand. Compcert-a formally verified optimizing compiler. In *ERTS 2016: Embedded Real Time Software and Systems, 8th European Congress*, 2016.

[Li *et al.*, 2019] Jianlin Li, Jiangchao Liu, Pengfei Yang, Liqian Chen, Xiaowei Huang, and Lijun Zhang. Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification. In Bor-Yuh Evan Chang, editor, *Static Analysis - 26th International Symposium, SAS 2019, Porto, Portugal, October 8-11, 2019, Proceedings*, volume 11822 of *Lecture Notes in Computer Science*, pages 296–319. Springer, 2019.

[Li *et al.*, 2020] Renjue Li, Jianlin Li, Cheng-Chao Huang, Pengfei Yang, Xiaowei Huang, Lijun Zhang, Bai Xue, and Holger Hermanns. PRODeep: a platform for robustness verification of deep neural networks. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, pages 1630–1634, New York, NY, USA, November 2020. Association for Computing Machinery.

[Liu *et al.*, 2019] Changliu Liu, Tomer Arnon, Christopher Lazarus, Clark Barrett, and Mykel J. Kochenderfer. Algorithms for Verifying Deep Neural Networks. *arXiv:1903.06758 [cs, stat]*, March 2019.

[Manfredi and Jestin, 2016] Guido Manfredi and Yannick Jestin. An introduction to ACAS Xu and the challenges

ahead. In *IEEE/AIAA Digital Avionics Systems Conference (DASC)*, Sacramento, CA, USA, September 2016.

[Ostrovsky *et al.*, 2022] Matan Ostrovsky, Clark W. Barrett, and Guy Katz. An abstraction-refinement approach to verifying convolutional neural networks. *CoRR*, abs/2201.01978, 2022.

[Palma *et al.*,] Alessandro De Palma, Rudy Bunel, Alban Desmaison, Krishnamurthy Dvijotham, Pushmeet Kohli, Philip H. S. Torr, and M. Pawan Kumar. Improved branch and bound for neural network verification via lagrangian decomposition.

[Ranzato and Zanella, 2019] Francesco Ranzato and Marco Zanella. Robustness verification of support vector machines. In *International Static Analysis Symposium*, pages 271–295. Springer, 2019.

[Shi *et al.*, 2020] Zhouxing Shi, Huan Zhang, Kai-Wei Chang, Minlie Huang, and Cho-Jui Hsieh. Robustness Verification for Transformers. In *International Conference on Learning Representations*, 2020.

[Shriver *et al.*, 2021] David Shriver, Sebastian Elbaum, and Matthew B. Dwyer. DNNV: A Framework for Deep Neural Network Verification. In Alexandra Silva and K. Rustan M. Leino, editors, *Computer Aided Verification*, Lecture Notes in Computer Science, pages 137–150, Cham, 2021. Springer International Publishing.

[Singh *et al.*, 2018] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and Effective Robustness Certification. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 10802–10813. Curran Associates, Inc., 2018.

[Singh *et al.*, 2019] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–30, 2019.

[Urban and Miné, 2021] Caterina Urban and Antoine Miné. A Review of Formal Methods applied to Machine Learning. *arXiv:2104.02466 [cs]*, April 2021.

[Urban *et al.*, 2019] Caterina Urban, Maria Christakis, Valentin Wüstholz, and Fuyuan Zhang. Perfectly Parallel Fairness Certification of Neural Networks. *arXiv:1912.02499 [cs]*, December 2019.

[Wang *et al.*,] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J. Zico Kolter. Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Complete and Incomplete Neural Network Robustness Verification.

[Wang *et al.*, 2018] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient Formal Safety Analysis of Neural Networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 6367–6377. Curran Associates, Inc., 2018.

[Wu *et al.*, 2022] Haoze Wu, Aleksandar Zeljić, Guy Katz, and Clark Barrett. Efficient Neural Network Analysis with Sum-of-Infeasibilities. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, pages 143–163, Cham, January 2022. Springer International Publishing.

[Xu *et al.*, 2021] 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. *arXiv:2011.13824 [cs]*, March 2021. arXiv: 2011.13824.