Languages and tools for formal verification

ESSAI 2025

Julien Girard-Satabin
Zakaria Chihani
Dorin Doncenco

CEA LIST

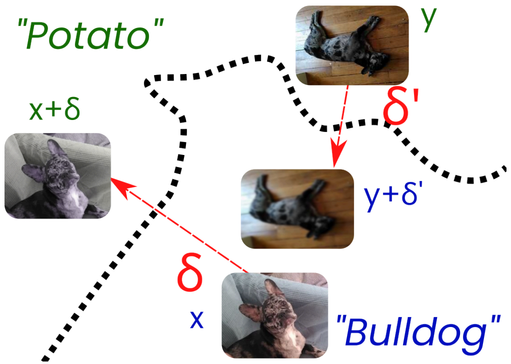
2025-07-04

This work was supported by the French Agence Nationale de la Recherche (ANR) through SAIF (ANR-23-PEIA-0006) and DeepGreen (ANR-23-DEGR-0001) as part of the France 2030 programme.

Summing up before going further

What we have seen so far

Local robustness

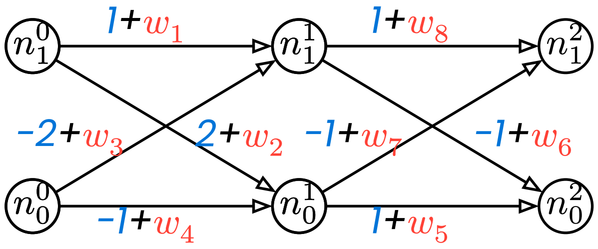


Checks that your network is correct and robust

Formal Explanations

Helps you understand how your model takes decisions

Testing and debugging



Helps you finding faulty inputs and correct the net

A due reminder

Local robustness [1]

Let a classifier $f : \mathcal{X} \mapsto \mathcal{Y}$. Given $x \in \mathcal{X}$ and $\varepsilon \in \mathbb{R} << 1$ the problem of *local robustness* is to prove that $\forall x^{\{'\}}.\ \|x - x^{\{'\}}\|_p < \varepsilon \rightarrow f(x) = f\left(x^{\{'\}}\right)$

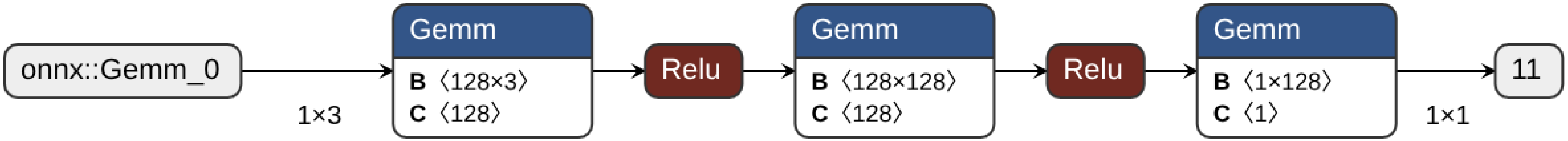
Sprinkled over the whole course and yet, we discussed very little on how it is actually encoded

Content of this last session

This final course will delve into practicalities of formal verification of neural networks

- tools
- languages
- social community and venues
- some future possible research tracks, informed by the past

Tools



A neural network can be represented as a directed acyclic graph (DAG)

ONNX

Representing neural networks

Open Neural Network eXchange (ONNX) format: 196 operators

ONNX Operators

Lists out all the ONNX operators. For each operator, lists out the usage guide, parameters, examples, and line-by-line version history. This section also includes tables detailing each operator with its versions, as done in Operators.md.

All examples end by calling function *expect*. which checks a runtime produces the expected output for this example. One implementation based on onnxruntime can be found at Sample operator test code.

ai.onnx | ai.onnx.ml | ai.onnx.preview.training

operator	versions	differences
Abs	13, 6, 1	13/6, 13/1, 6/1
Acos	22, 7	22/7
Acosh	22, 9	22/9
Add	14, 13, 7, 6, 1	14/13, 14/7, 13/7, 14/6, 13/6, 7/6, 14/1, 13/1, 7/1, 6/1
AffineGrid	20	
And	7, 1	7/1
ArgMax	13, 12, 11, 1	13/12, 13/11, 12/11, 13/1, 12/1, 11/1
ArgMin	13, 12, 11, 1	13/12, 13/11, 12/11, 13/1,

Conv

Conv - 22

Version

- **name**: Conv (GitHub)
- **domain**: `main`
- **since_version**: `22`
- **function**: `False`
- **support_level**: `SupportType.COMMON`
- **shape inference**: `True`

This version of the operator has been available **since version 22**.

Summary

The convolution operator consumes an input tensor and a filter, and computes the output.

Marabou (Complete SMT Solver)

GitHub repo

Marabou (successor of ReLUPlex [1]) is still actively developped [2], [3]

Actually the backend of most of Session 4 formal verification example

Marabou (Complete SMT Solver)

Core features

- A sound and complete reasoning engine, based on SMT calculus (see Session 2.)
- Support advanced checking techniques:
  - Proof productions [4] and certificates [5]
  - Parallel verification with Divide and Conquer

PyRAT (Abstract Interpretation Solver)

PyRAT

Abstract-interpretation based analyzer developped by our team [6], used in several real-world application [7]

DEEPGREEN

Confiance.ai

Fancy demo

Freely available for academic purpose

PyRAT (Abstract Interpretation Solver)

Core features

- Vastest ONNX support among verifiers
- Support for state-of-the-art abstract interpretation domains
  - all the zonotopes variants defined in session 2!
- *Soundness* mode with regards to real-value arithmetic
- Fast counterexample search with adversarial attacks
- Branch and bound approaches for complete mode

$\alpha - \beta - \mathrm{CROWN}$ (Abstract Interpretation Solver)

CROWN

Winner of International Verification of Neural Networks Competitions (VNN-COMP 2021 - 2024)

$\alpha - \beta - \mathrm{CROWN}$ [8] consistently wins VNN-Comp since 2021

Other tools

- NNV [9]
- nnenum [10]
- Saver [11]
- NeuralSAT [12]
- MIPVerify [13]

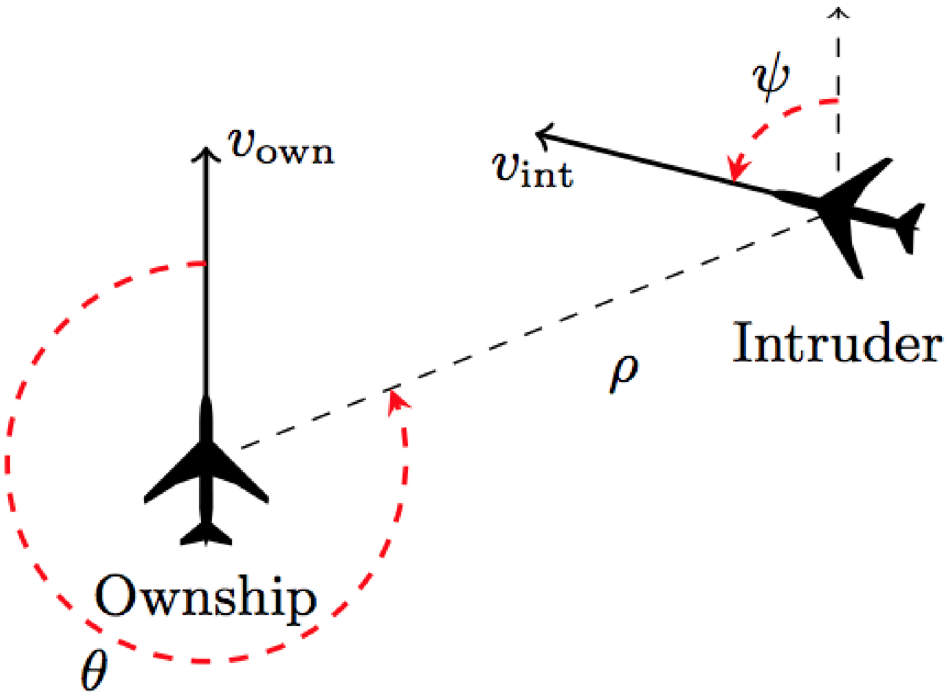
For more details, see [14]

Some observations

- Survivors of an initial cambrian explosion of tools (started my PhD in 2017, there was no one)
- Tools were initially specialized into a single technique, now everybody does (some flavour of) abstract interpretation and everybody has (some flavour of) completeness

Evaluating those tools

The initial benchmark: ACAS-Xu



Evaluating those tools

But then rose several questions:

- beyond linear and convolutional layers (skip connections?)
- deeper neural networks

The International Verification of Neural Network Competition (VNN-Comp)

The 5th International Verification of Neural Networks Competition (VNN-COMP 2024): Summary and Results

Christopher Brix[1], Stanley Bak[2], Taylor T. Johnson[3], and Haoze Wu[4]

[1] RWTH Aachen University, Aachen, Germany
brix@cs.rwth-aachen.de
[2] Stony Brook University, Stony Brook, New York, USA
stanley.bak@stonybrook.edu
[3] Vanderbilt University, Nashville, Tennessee, USA
taylor.johnson@vanderbilt.edu
[4] Amherst College, Amherst, Massachussett, USA
hwu@amherst.edu

The International Verification of Neural Network Competition (VNN-Comp)

Visit VNN-Comp website and *skim through last year report* and maybe the actual results on the *github repo*?

Organized by Christopher Brix, Stanley Bak, Taylor T. Johnson, and Haoze Wu (shout outs!!) for 2021 onward

The International Verification of Neural Network Competition (VNN-Comp)

- 16 different benchmarks, comprising properties and neural network to verify
- each year: a phase of collegial discussion on the rules of the competition
- improvements and refinements on the scoring, various tracks, new contenders…

On tools disagreements

Property	Marabou				marabоupy				PyRAT				nnenum			
	T_n	A_n	T_u	A_u	T_n	A_n	T_u	A_u	T_n	A_n	T_u	A_u	T_n	A_n	T_u	A_u
ϕ_1	3.00	(?)	3.00	(?)	5.00	(✓)	243.00	(⏱)	8.00	(✓)	11.00	(✓)	4.00	(✓)	4.00	(✓)
ϕ_2	37.00	(✓)	3.00	(?)	26.00	(✓)	243.00	(⏱)	19.00	(✓)	38.00	(✓)	4.00	(✓)	4.00	(✓)
ϕ_3	243.00	(⏱)	5.00	(?)	243.00	(⏱)	243.00	(⏱)	246.00	(⏱)	246.00	(⏱)	4.00	(✓)	4.00	(✓)
ϕ_4	44.00	(✓)	5.00	(?)	36.00	(✓)	4.00	(✗)	25.00	(✓)	246.00	(⏱)	4.00	(✓)	4.00	(✓)
ϕ_5	102.00	(✓)	5.00	(?)	93.00	(✓)	5.00	(✗)	246.00	(⏱)	246.00	(⏱)	4.00	(✓)	5.00	(✓)
ϕ_6	558.00	(✓)	5.00	(?)	566.00	(✓)	1925.00	(⏱)	156.00	(✓)	426.00	(⏱)	7.00	(✓)	13.00	(?)
ϕ_7	485.00	(⏱)	5.00	(?)	484.00	(⏱)	484.00	(⏱)	246.00	(⏱)	246.00	(⏱)	119.00	(✗)	4.00	(?)
ϕ_8	485.00	(✗)	5.00	(?)	8.00	(✗)	248.00	(⏱)	246.00	(⏱)	246.00	(⏱)	4.00	(✗)	4.00	(✗)
ϕ_9	182.00	(✓)	5.00	(?)	222.00	(✓)	5.00	(✗)	61.00	(✓)	246.00	(⏱)	6.00	(✓)	9.00	(✓)
ϕ_{10}	83.00	(✓)	3.00	(?)	151.00	(✗)	245.00	(✗)	13.00	(✓)	246.00	(⏱)	4.00	(✓)	5.00	(✓)

Limitations

- Soundness of provers with floating point arithmetic does not yet exist [15]
- Still existing bugs [16]
- Some provers are difficult to install because the Python packaging ecosystem being what it is

So should I abandon all hope?

No!

SAT and SMT solvers that are now used have **decades** of work put on their soundness and their quality

Languages

VNN-Lib

VNN-Lib [17] is the *de-facto* standard for the International Competition of Neural Network Verification (VNN-Comp [18], [19], [20])

It is a subset of SMTLIB [21], classical specification language for SMT calculus (more specifically, the theory of **Quantifier-Free Linear Real Arithmetic** `QF_LRA`)

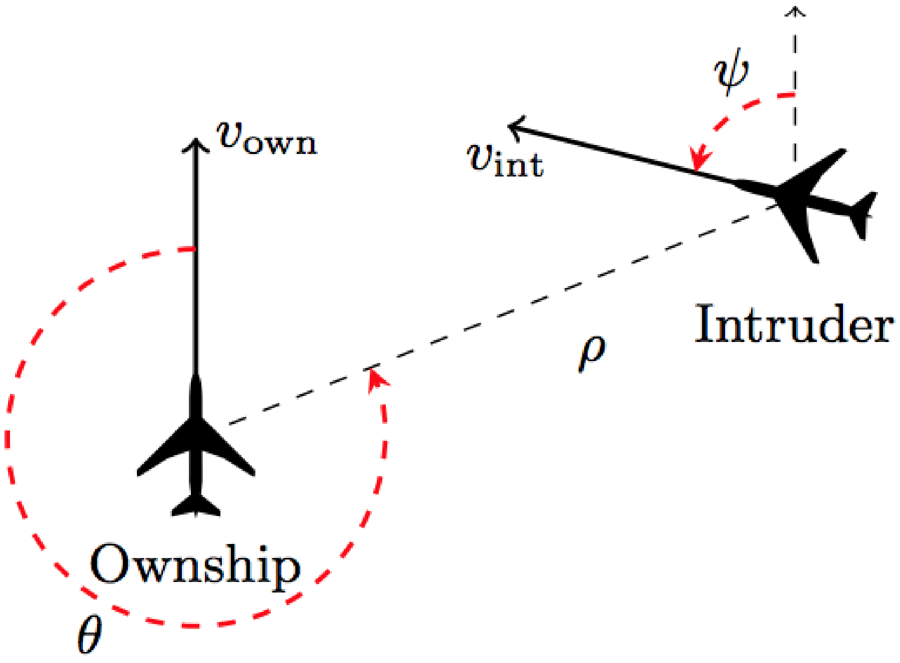
VNN-Lib

VNN-Lib [17] is the *de-facto* standard for the International Competition of Neural Network Verification (VNN-Comp [18], [19], [20])

It is a subset of SMTLIB [21], classical specification language for SMT calculus (more specifically, the theory of **Quantifier-Free Linear Real Arithmetic** `QF_LRA`)

- **Quantifier-Free**: no universal quantification ∀: everything must be existentially quantified
- **Linear**: only linear operations allowed between variables
- **Real Arithmetic**: computations are expected to be on Real (in practice, Rational) numbers

ACAS-Xu specification in VNN-Lib



Property ϕ_1.

– Description: If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will always be below a certain fixed threshold.
– Tested on: all 45 networks.
– Input constraints: $\rho \geq 55947.691$, $v_{\text{own}} \geq 1145$, $v_{\text{int}} \leq 60$.
– Desired output property: the score for COC is at most 1500.

Property ϕ_2.

– Description: If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will never be maximal.
– Tested on: $N_{x,y}$ for all $x \geq 2$ and for all y.
– Input constraints: $\rho \geq 55947.691$, $v_{\text{own}} \geq 1145$, $v_{\text{int}} \leq 60$.
– Desired output property: the score for COC is not the maximal score.

Property ϕ_3.

– Description: If the intruder is directly ahead and is moving towards the ownship, the score for COC will not be minimal.
– Tested on: all networks except $N_{1,7}$, $N_{1,8}$, and $N_{1,9}$.
– Input constraints: $1500 \leq \rho \leq 1800$, $-0.06 \leq \theta \leq 0.06$, $\psi \geq 3.10$, $v_{\text{own}} \geq 980$, $v_{\text{int}} \geq 960$.
– Desired output property: the score for COC is not the minimal score.

ACAS-Xu specification in VNN-Lib

```
(declare-const X_0 Real)
(declare-const X_1 Real)
(declare-const X_2 Real)
(declare-const X_3 Real)
(declare-const X_4 Real)

(declare-const Y_0 Real)
(declare-const Y_1 Real)
(declare-const Y_2 Real)
(declare-const Y_3 Real)
(declare-const Y_4 Real)

; Unscaled Input 0: (36000, 60760)
(assert (<= X_0 0.679857769))
(assert (>= X_0 0.268978427))

; Unscaled Input 1: (0.7, 3.141592)
(assert (<= X_1 0.499999896))
(assert (>= X_1 0.11140846))
```

```
; Unscaled Input 2: (-3.141592, -3.1315920000000004)
(assert (<= X_2 -0.498408347))
(assert (>= X_2 -0.499999896))

; Unscaled Input 3: (900, 1200)
(assert (<= X_3 0.5))
(assert (>= X_3 0.227272727))

; Unscaled Input 4: (600, 1200)
(assert (<= X_4 0.5))
(assert (>= X_4 0.0))

; unsafe if coc is not minimal
(assert (or
    (and (<= Y_1 Y_0))
    (and (<= Y_2 Y_0))
    (and (<= Y_3 Y_0))
    (and (<= Y_4 Y_0))
))
```

Something is missing, right?

ACAS-Xu specification in VNN-Lib

```
(declare-const X_0 Real)
(declare-const X_1 Real)
(declare-const X_2 Real)
(declare-const X_3 Real)
(declare-const X_4 Real)

(declare-const Y_0 Real)
(declare-const Y_1 Real)
(declare-const Y_2 Real)
(declare-const Y_3 Real)
(declare-const Y_4 Real)

; Unscaled Input 0: (36000, 60760)
(assert (<= X_0 0.679857769))
(assert (>= X_0 0.268978427))

; Unscaled Input 1: (0.7, 3.141592)
(assert (<= X_1 0.499999896))
(assert (>= X_1 0.11140846))
```

```
; Unscaled Input 2: (-3.141592, -3.1315920000000004)
(assert (<= X_2 -0.498408347))
(assert (>= X_2 -0.499999896))

; Unscaled Input 3: (900, 1200)
(assert (<= X_3 0.5))
(assert (>= X_3 0.227272727))

; Unscaled Input 4: (600, 1200)
(assert (<= X_4 0.5))
(assert (>= X_4 0.0))

; unsafe if coc is not minimal
(assert (or
    (and (<= Y_1 Y_0))
    (and (<= Y_2 Y_0))
    (and (<= Y_3 Y_0))
    (and (<= Y_4 Y_0))
))
```

Something is missing, right? Which network are we verifying??

Limitations of VNNLib

- Does not specify anything regarding the neural network
- Specification size is linear in the size of the input: good luck proofreading this :)
- Does not represent actual computations (real arithmetic)

Limitations of VNNLib

Given $\mathrm{nn}_1, \mathrm{nn}_2, x \in \mathbb{R}^2, \varepsilon \in \mathbb{R} \lll 1$ and $H(x_0, x_1, \varepsilon)$ a set of hypotheses

Let the formula $\forall x_0, x_1, \varepsilon. H(x_0, x_1, \varepsilon) \Rightarrow \mathrm{nn}_2(\mathrm{nn}_1, (x_1), x_1 + \varepsilon) + \mathrm{nn}_1(x_0) > 0$

Limitations of VNNLib

Given $\mathrm{nn}_1, \mathrm{nn}_2, x \in \mathbb{R}^2, \varepsilon \in \mathbb{R} \lll 1$ and $H(x_0, x_1, \varepsilon)$ a set of hypotheses

Let the formula $\forall x_0, x_1, \varepsilon . H(x_0, x_1, \varepsilon) \Rightarrow \mathrm{nn}_2(\mathrm{nn}_1, (x_1), x_1 + \varepsilon) + \mathrm{nn}_1(x_0) > 0$

This property is **not** amenable for provers winners of the VNN-Competition

Limitations of VNNLib

Given $\mathrm{nn}_1, \mathrm{nn}_2, x \in \mathbb{R}^2, \varepsilon \in \mathbb{R} \lll 1$ and $H(x_0, x_1, \varepsilon)$ a set of hypotheses

Let the formula $\forall x_0, x_1, \varepsilon . H(x_0, x_1, \varepsilon) \Rightarrow \mathrm{nn}_2(\mathrm{nn}_1, (x_1), x_1 + \varepsilon) + \mathrm{nn}_1(x_0) > 0$

This property is **not** amenable for provers winners of the VNN-Competition

$$\forall x_0, x_1, \varepsilon . H(x_0, x_1, \varepsilon) \Rightarrow$$

$$\left.\overbrace{\underbrace{\mathrm{nn}_2(\mathrm{nn}_1}_{\text{Multiple NNs}}, (x_1), \underbrace{x_1 + \varepsilon}_{\text{Operation on inputs}}}^{\text{Composition of NN}}) + \mathrm{nn}_1(x_0) > 0 \right\} \text{Comparison of outputs}$$

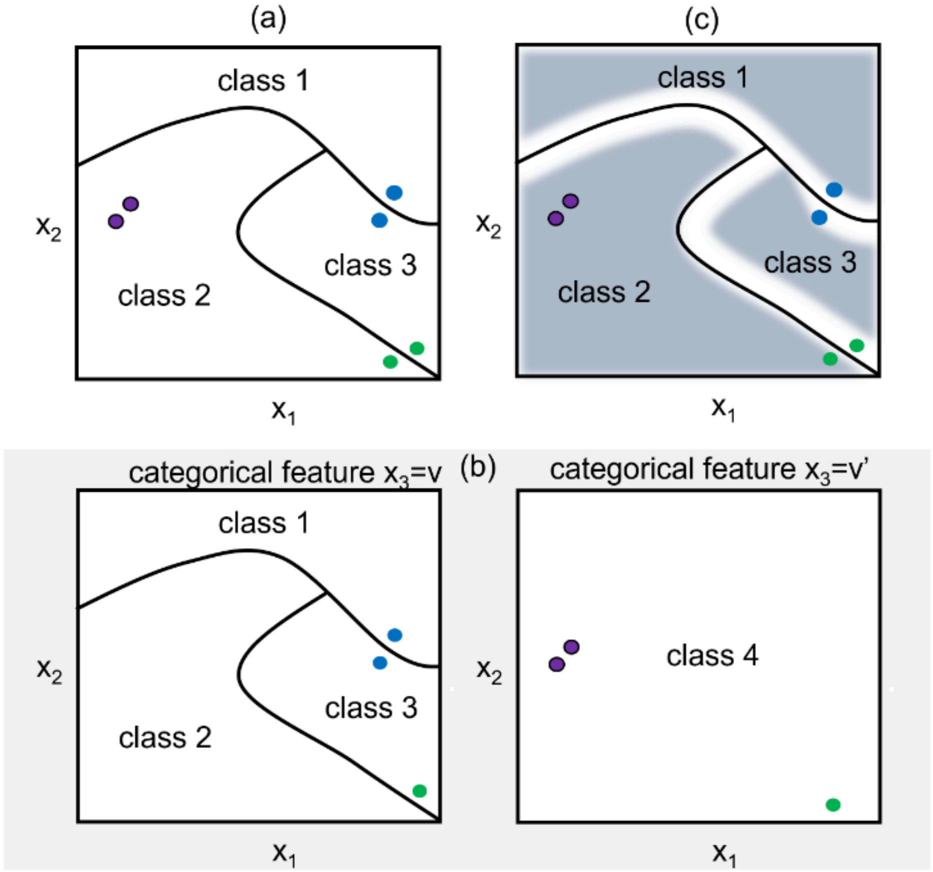
Limitations of VNNLib

Confidence-based robustness [22]

$$\forall x, x', \text{cond}(x, x', \varepsilon) \wedge \text{conf}(f(x)) > \kappa \Rightarrow \text{class}(f(x)) = \text{class}(f(x'))$$

For all couple of inputs, as long as the network is confident enough in its prediction, prediction should not change

And a whole family of *hyperproperties* (multiple execution traces)



Frustrations to be adressed

- Inaccurate specification language
- No clear way to derive higher-order formulas to VNN-Lib
- Collection of tools that are difficult to install and compare

Frustrations to be adressed with CAISAR

CAISAR

A specification language *and* a set of tools to ease formal verification [23]

Free and Open-Source Software with a dedicated manual https://caisar-platform.com

A richer specification language

```
⟨decl⟩    type ⟨tId⟩ = ⟨type⟩
        | predicate ⟨id⟩
             ⟨binder⟩* = ⟨expr⟩
        | function ⟨id⟩
             ⟨binder⟩* ⟨spec⟩* = ⟨expr⟩

⟨type⟩    ⟨tId⟩
        | ⟨type⟩ → ⟨type⟩
        | (⟨type⟩,... ,⟨type⟩)
        | vector ⟨type⟩
        | int | bool | float | string
        | model

⟨binder⟩  ⟨id⟩ | (⟨id⟩ : ⟨type⟩)

⟨spec⟩    requires {⟨expr⟩}
        | ensures {⟨expr⟩}

⟨bop⟩     ≤ | ≥ | < | >
        | + | − | × | /
        | ∧ | ∨ | →

⟨expr⟩    ⟨id⟩
        | ⟨built-in⟩
        | ⟨expr⟩⟨expr⟩
        | (⟨expr⟩,... ,⟨expr⟩)
        | let ⟨id⟩ = ⟨expr⟩ in
        | if ⟨expr⟩ then ⟨expr⟩
          else ⟨expr⟩
        | ⟨expr⟩⟨bop⟩⟨expr⟩
        | forall⟨binder⟩.⟨expr⟩
        | exists⟨binder⟩.⟨expr⟩
        | not⟨expr⟩
        | i ∈ Integer
        | {true, false} ∈ Boolean
        | f ∈ Float | s ∈ String

⟨built-in⟩  read_model ⟨expr⟩
          | length ⟨expr⟩
          | has_length ⟨expr⟩ ⟨expr⟩
          | ⟨expr⟩[⟨expr⟩]
          | ⟨expr⟩@@⟨expr⟩
```

CAISAR specification language. Quantifier are partially supported.

A richer specification language

Time to *read a local robustness specification on MNIST!*

A richer specification language

```
theory MNIST

  use ieee_float.Float64
  use caisar.types.Float64WithBounds as Feature
  use caisar.types.IntWithBounds as Label
  use caisar.model.Model

  use caisar.dataset.CSV
  use caisar.robust.ClassRobustCSV

  constant model_filename: string
  constant dataset_filename: string
  [...]
```

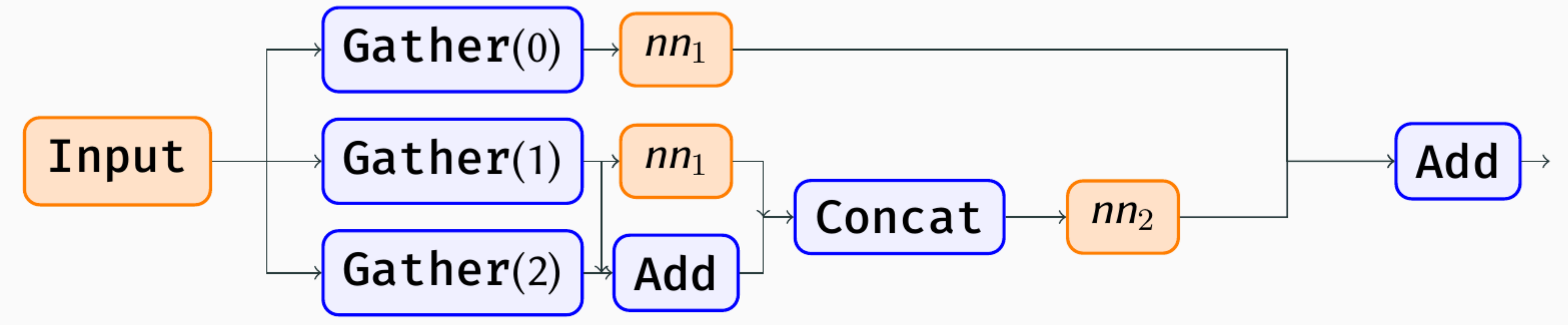
```
  [...]
  constant label_bounds: Label.bounds =
    Label.{ lower = 0; upper = 9 }

  constant feature_bounds: Feature.bounds =
    Feature.{ lower = (0.0:t); upper = (1.0:t) }

  goal robustness:
    let nn = read_model model_filename in
    let dataset = read_dataset dataset_filename in
    let eps = (0.125:t) in (* Need to represent floats explicitly *)
    robust feature_bounds label_bounds nn dataset eps
end
```

A richer specification language

Integrates an automated graph editing technique to integrate specifications inside of the control-flow, *à la* neurosymbolic



Gather(0) (resp. **Gather**(1)) extracts x_0 (resp. x_1) and **Gather**(2) extracts ϵ from the **Input** node. First **Add** computes $x_1 + \epsilon$. Nodes nn_1 nn_2 are the inlined nn_1 and nn_2 control flows. **Concat** prepares nn_2 inputs.

Vehicle

```
type Image = Tensor Rat [28, 28]
type Label = Index 10

validImage : Image -> Bool
validImage x = forall i j . 0 <= x ! i ! j
<= 1

@network
classifier : Image -> Vector Rat 10

advises : Image -> Label -> Bool
advises x i = forall j . j != i =>
classifier x ! i > classifier x ! j

@parameter
epsilon : Rat
```

```
boundedByEpsilon : Image -> Bool
boundedByEpsilon x = forall i j . -epsilon
<= x ! i ! j <= epsilon

robustAround : Image -> Label -> Bool
robustAround image label = forall
pertubation .
  let perturbedImage = image - pertubation
in
  boundedByEpsilon pertubation and
validImage perturbedImage =>
    advises perturbedImage label

@parameter(infer=True)
n : Nat
```

```
@dataset
trainingImages : Vector Image n

@dataset
trainingLabels : Vector Label n

@property
robust : Vector Bool n
robust = foreach i . robustAround
(trainingImages ! i) (trainingLabels ! i)
```

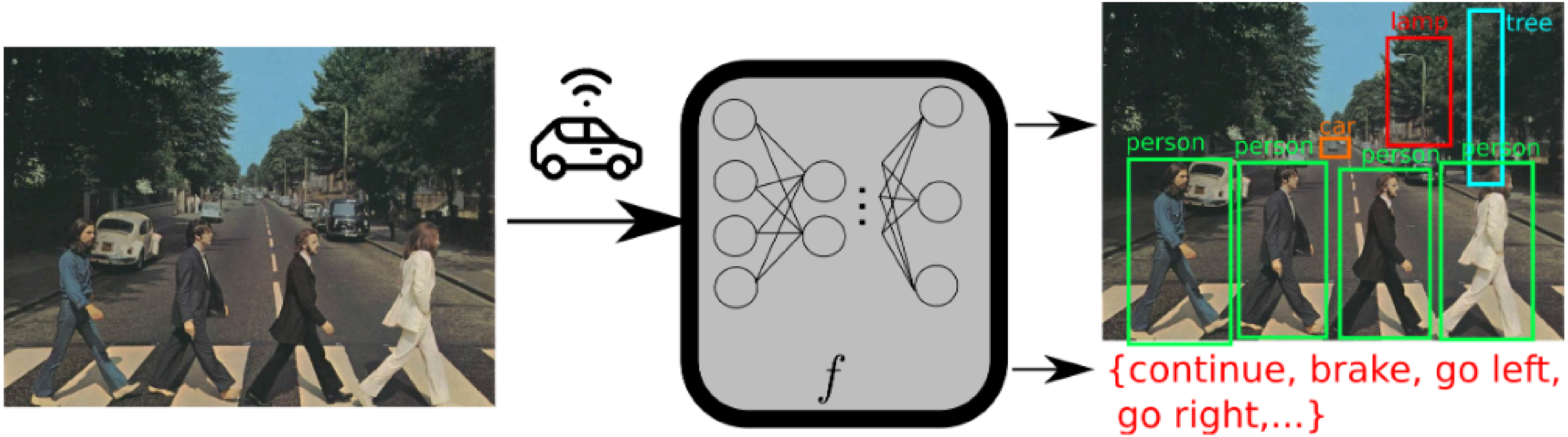
A higher-level specification language [24], [25]. Displayed here is the full Vehicle specification for MNIST robustness

Support of numerous provers

- 9 provers supported (including all VNN-Comp winners)
- reproducible build and experiments thanks to the Nix package manager

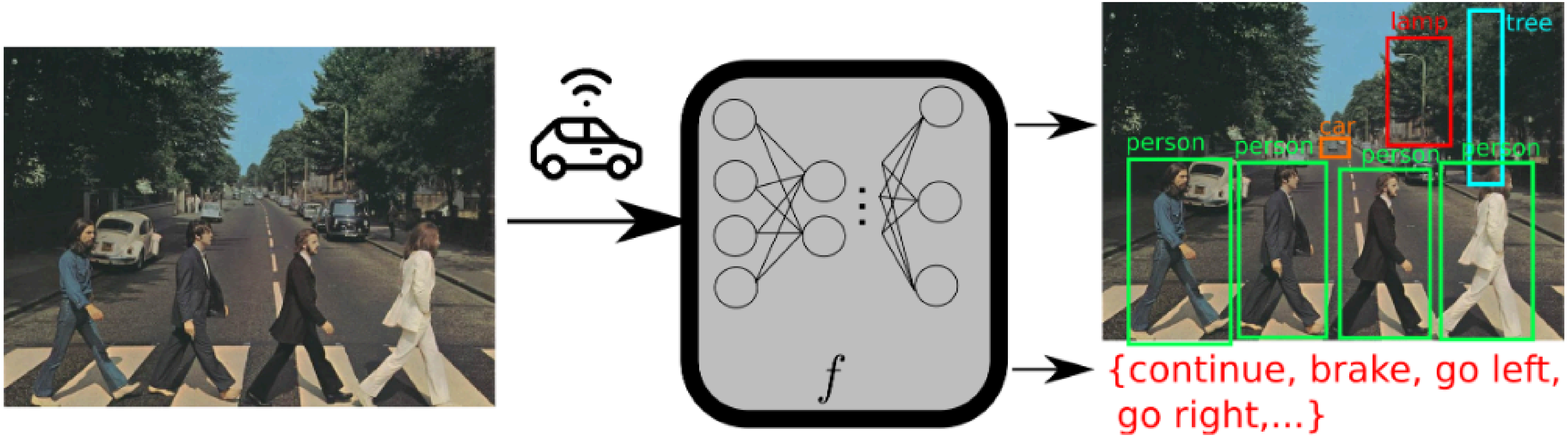
- a repository of examples and (soon) benchmarks from the VNN-Comp

On the specification problem



To verify that system, one first needs to define its inputs

$$\forall x . x \in \{\text{image with pedestrian}\} \Rightarrow f(x) = \text{brake}$$



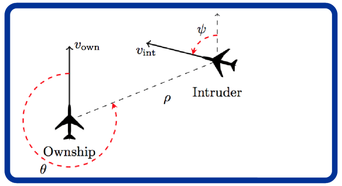
To verify that system, one first needs to define its inputs

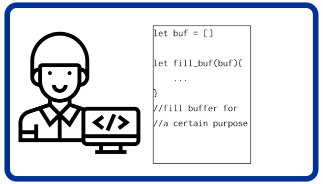
$$\forall x. x \in \{\text{image with pedestrian}\} \Rightarrow f(x) = \text{brake}$$

What is an image containing a pedestrian? How to *specify* it?

What makes machine learning hard to verify

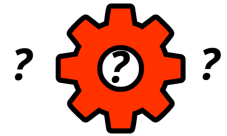
	Specification languages	Writer agent	Verification tooling
Classical programs	*Explicit specification, derived from requirements*	*A team of humans directly writes the program*	*Mature tools*
Neural networks	*High dimensional input, data as partial specification*	*A loss minimization scheme parametrizes the program*	*Still experimental*





Astrée Software, PolySpace, Compcert, ESTEREL Technologies, frama-c, z3 Research, Software Analyzers, atelier B

$$\min_\theta \mathcal{L}(\theta)$$



What makes machine learning hard to verify

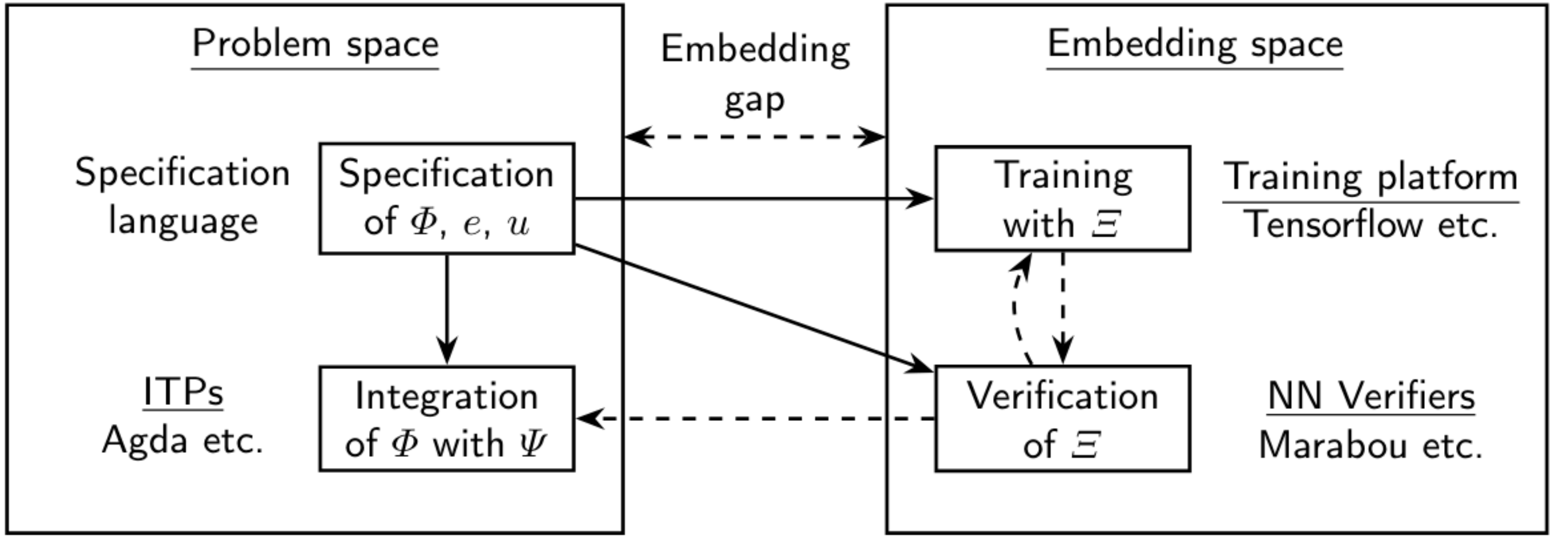
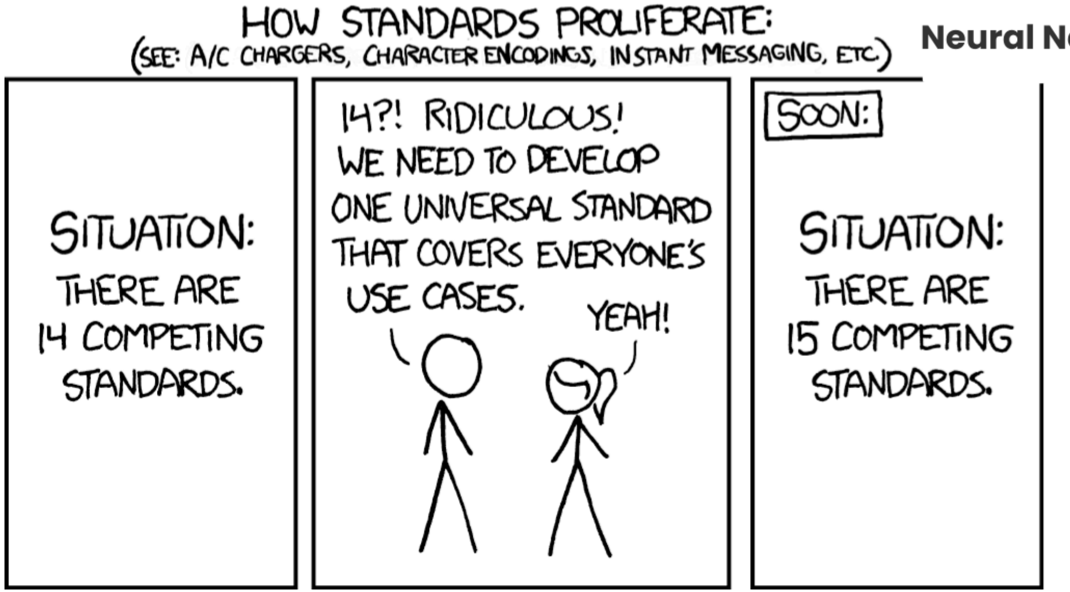


Fig. 6: *Outline of Vehicle compiler backends, bridging the Embedding Gap [33,32]. Dashed lines indicate information flow and solid lines automatic compilation.*

The *embedding gap* we describe [26]

Neural Network specification languages



Adapted from Randall Munroe

Closing remarks on the course

Discussion

Specification languages

- Exploring neuro-symbolic specification using simulators/generators [27], [28]
- Closing the embedding gap as much as possible
- Refining higher-order specification into concrete verification / constraints

Tools

- Ensuring actual soundness of the tool is paramount
- Debug! In a cool way!
- Automate tool configuration ?

Discussion

Community

- Help organize the VNN-Comp!
- Propose use cases (Graph Neural Networks?)
- Aim towards other applications!
- Existing venues are growing (AISafety, SafeComp, workshops in AI/ML AND Verification Conferences)

Feedback form

You did well and learnt a lot of things!

Feedback form: *https://framaforms.org/feedback-on-essai-course-formal-verification-of-symbolic-and-connectionist-ai-a-way-toward-higher*

Bibliography

[1] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks," in *Computer Aided Verification*, Springer International Publishing, 2017, pp. 97–117. doi: [10.1007/978-3-319-63387-9_5](https://doi.org/10.1007/978-3-319-63387-9_5).

[2] G. Katz *et al.*, "The Marabou Framework for Verification and Analysis of Deep Neural Networks," *Computer Aided Verification*, vol. 11561. Springer International Publishing, Cham, pp. 443–452, 2019. doi: [10.1007/978-3-030-25540-4_26](https://doi.org/10.1007/978-3-030-25540-4_26).

[3] H. Wu *et al.*, "Marabou 2.0: A Versatile Formal Analyzer of Neural Networks." arXiv, 2024. doi: [10.48550/ARXIV.2401.14461](https://doi.org/10.48550/ARXIV.2401.14461).

[4] O. Isac, C. Barrett, M. Zhang, and G. Katz, "Neural Network Verification with Proof Production." [Online]. Available: https://arxiv.org/abs/2206.00512

[5] R. Desmartin, O. Isac, G. Passmore, E. Komendantskaya, K. Stark, and G. Katz, "A Certified Proof Checker for Deep Neural Network Verification in Imandra." [Online]. Available: https://arxiv.org/abs/2405.10611

[6] A. Lemesle, J. Lehmann, and T. L. Gall, "Neural Network Verification with PyRAT." arXiv, 2024. doi: [10.48550/ARXIV.2410.23903](https://doi.org/10.48550/ARXIV.2410.23903).

[7] C. Gabreau *et al.*, "A study of an ACAS-Xu exact implementation using ED-324/ARP6983," in *12th European Congress Embedded Real Time Systems - ERTS 2024*, Toulouse (31000), France, Jun. 2024. [Online]. Available: https://hal.science/hal-04584782

[8] S. Wang *et al.*, "Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Complete and Incomplete Neural Network Robustness Verification." Accessed: Mar. 04, 2022. [Online]. Available: http://arxiv.org/abs/2103.06624

[9] D. M. Lopez, S. W. Choi, H.-D. Tran, and T. T. Johnson, "NNV 2.0: The Neural Network Verification Tool," in *Computer Aided Verification*, C. Enea and A. Lal, Eds., Cham: Springer Nature Switzerland, 2023, pp. 397–412.

[10] S. Bak, "Nnenum: Verification of ReLU Neural Networks with Optimized Abstraction Refinement," in *NASA Formal Methods*, A. Dutle, M. M. Moscato, L. Titolo, C. A. Muñoz, and I.

Perez, Eds., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 19–36. doi: 10.1007/978-3-030-76384-8_2.

[11] F. Ranzato and M. Zanella, "Robustness Verification of Support Vector Machines," in *Static Analysis*, B.-Y. E. Chang, Ed., Cham: Springer International Publishing, 2019, pp. 271–295.

[12] H. Duong, L. Li, T. Nguyen, and M. Dwyer, "A DPLL(T) Framework for Verifying Deep Neural Networks." 2023.

[13] V. Tjeng, K. Xiao, and R. Tedrake, "Evaluating Robustness of Neural Networks with Mixed Integer Programming," presented at the International Conference on Learning Representations (ICLR), 2019. Accessed: Jun. 19, 2019. [Online]. Available: https://openreview.net/pdf?id=HyGIdiRqtm

[14] C. Urban and A. Miné, "A Review of Formal Methods Applied to Machine Learning," *arXiv:2104.02466 [cs]*, Apr. 2021.

[15] D. Zombori, B. Bánhelyi, T. Csendes, I. Megyeri, and M. Jelasity, "Fooling a Complete Neural Network Verifier," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=4IwieFS44l

[16] R. Elsaleh and G. Katz, "DelBugV: Delta-Debugging Neural Network Verifiers." arXiv, 2023. doi: [10.48550/ARXIV.2305.18558](https://doi.org/10.48550/ARXIV.2305.18558).

[17] S. Demarchi, D. Guidotti, L. Pulina, and A. Tacchella, "Supporting Standardization of Neural Networks Verification with VNN-LIB and CoCoNet.," in *6th Workshop on Formal Methods for ML-Enabled Autonomous Systems*, Jul. 2023.

[18] C. Brix, M. N. Müller, S. Bak, T. T. Johnson, and C. Liu, "First Three Years of the International Verification of Neural Networks Competition (VNN-COMP)." 2023.

[19] C. Brix, S. Bak, C. Liu, and T. T. Johnson, "The Fourth International Verification of Neural Networks Competition (VNN-COMP 2023): Summary and Results." 2023.

[20] C. Brix, S. Bak, T. T. Johnson, and H. Wu, “The Fifth International Verification of Neural Networks Competition (VNN-COMP 2024): Summary and Results.” [Online]. Available: https://arxiv.org/abs/2412.19985

[21] C. Barrett, P. Fontaine, and C. Tinelli, “The Satisfiability Modulo Theories Library (SMT-LIB).” 2016.

[22] A. Athavale, E. Bartocci, M. Christakis, M. Maffei, D. Nickovic, and G. Weissenbacher, “Verifying Global Two-Safety Properties in Neural Networks with Confidence,” in *Computer Aided Verification*, A. Gurfinkel and V. Ganesh, Eds., Cham: Springer Nature Switzerland, Jun. 2024, pp. 329–351. doi: *10.48550/arXiv.2405.14400*.

[23] M. Alberti, F. Bobot, Z. Chihani, J. Girard-Satabin, and A. Lemesle, “CAISAR: A platform for Characterizing Artificial Intelligence Safety and Robustness,” in *AISafety*, in CEUR-Workshop Proceedings. Vienne, Austria, Jul. 2022. [Online]. Available: https://hal.archives-ouvertes.fr/hal-03687211

[24] M. L. Daggitt, W. Kokke, R. Atkey, L. Arnaboldi, and E. Komendantskya, "Vehicle: Interfacing Neural Network Verifiers with Interactive Theorem Provers." [Online]. Available: https://arxiv.org/abs/2202.05207

[25] M. L. Daggitt, W. Kokke, R. Atkey, N. Slusarz, L. Arnaboldi, and E. Komendantskaya, "Vehicle: Bridging the Embedding Gap in the Verification of Neuro-Symbolic Programs." arXiv, 2024. doi: 10.48550/ARXIV.2401.06379.

[26] L. C. Cordeiro *et al.*, "Neural Network Verification is a Programming Language Challenge," 2025, doi: 10.48550/ARXIV.2501.05867.

[27] J. Girard-Satabin, G. Charpiat, Z. Chihani, and M. Schoenauer, "CAMUS: A Framework to Build Formal Specifications for Deep Perception Systems Using Simulators," in *ECAI 2020 - 24th European Conference on Artificial Intelligence*, Santiago de Compostela, Spain, Jun. 2020. [Online]. Available: https://hal.inria.fr/hal-02440520

[28] X. Xie, K. Kersting, and D. Neider, "Neuro-Symbolic Verification of Deep Neural Networks," in *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence,*

IJCAI-22, L. D. Raedt, Ed., International Joint Conferences on Artificial Intelligence Organization, 2022, pp. 3622–3628. doi: 10.24963/ijcai.2022/503.