Neural Network Verification is a Programming Language Challenge

Lucas C. Cordeiro[1] Matthew L. Daggitt[2] Julien Girard-Satabin[3] Omri Isac[4] Taylor T. Johnson[5] Guy Katz[4] Ekaterina Komendantskaya[6,7] Augustin Lemesle[3] **Edoardo Manino**[1] Artjoms Šinkarovs[6] Haoze Wu[8]

[1]University of Manchester, UK
[2]University of Western Australia, Australia
[3]Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France
[4]Hebrew University of Jerusalem, Israel
[5]Vanderbilt University, USA
[6]Southampton University, UK
[7]Heriot-Watt University, UK
[8]Amherst College, USA

ESOP - 7 May 2025

The textbook ML workflow

Three ingredients

- **Property.** Expressed as a **loss function**
- **Training.** Empirical risk minimisation over **finite data**
- **Implementation.** ML frameworks, HW accelerators

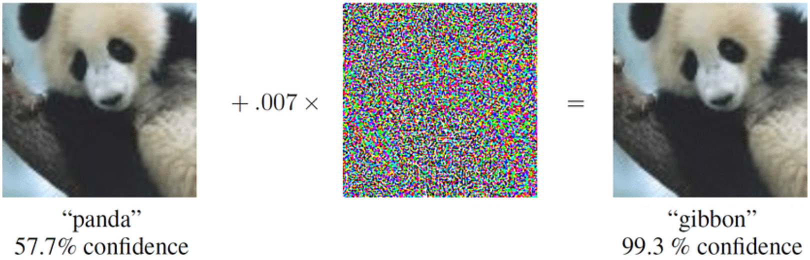
The textbook ML workflow

NN Property (optimisation objective) → NN Training → Original NN **Implementation**

Three ingredients

- **Property.** Expressed as a **loss function**
- **Training.** Empirical risk minimisation over **finite data**
- **Implementation.** ML frameworks, HW accelerators

Adversarial attacks



Adversarial attacks

Vision

- "Please make the neural network **safe**"

Adversarial attacks

Local robustness property

- For each image x in the training set
- Define a small local set of perturbations $||x - x'|| \leq \varepsilon$
- Ensure output class is $\arg\max\{f(x)\} = \arg\max\{f(x')\}$

Adversarial attacks

Programming languages interpretation

- Refinement type for f with predicate
- $\forall x, ||x - x'|| \leq \varepsilon \implies ||f(x) - f(x')|| \leq \delta$

Other “desirable” properties

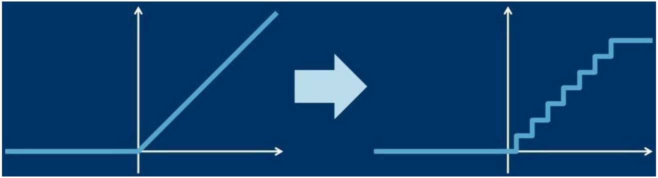
A high-level taxonomy

- **Geometric properties.** Defined along the data manifold. Examples: local robustness and equivalence, fairness.

Other “desirable” properties

A high-level taxonomy

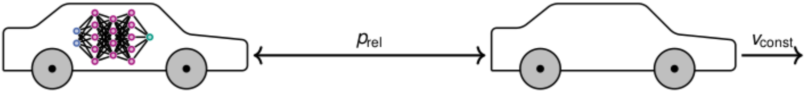
- **Geometric properties.** Defined along the data manifold.
  Examples: local robustness and equivalence, fairness.
- **Hyper-properties.** Must hold for any inputs.
  Examples: global robustness and equivalence, monotonicity.



Other “desirable” properties

A high-level taxonomy

- **Geometric properties.** Defined along the data manifold.
  Examples: local robustness and equivalence, fairness.
- **Hyper-properties.** Must hold for any inputs.
  Examples: global robustness and equivalence, monotonicity.
- **Domain specific.** Based on semantic of ML task.
  Examples: arbitrary pre- and post-conditions.

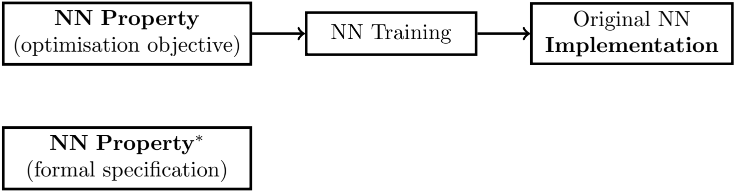


Neural network verification

Vision (v2)

- “Please **check** whether the neural network is safe”

Neural network verification



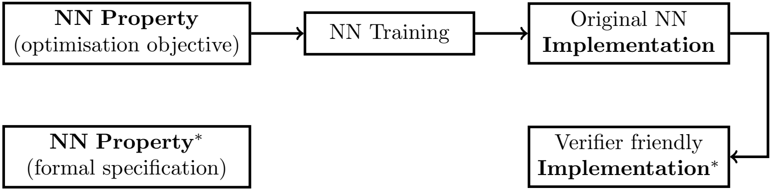
Vision (v2)

- "Please **check** whether the neural network is safe"

Solution

- **Property*.** Define what "safe" means.

Neural network verification



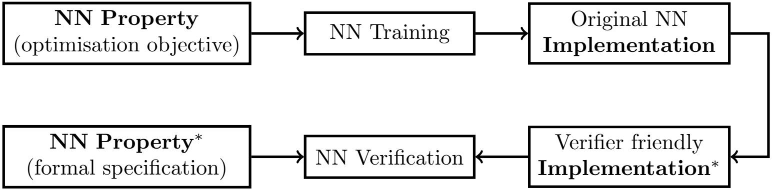
Vision (v2)

- "Please **check** whether the neural network is safe"

Solution

- **Property*.** Define what "safe" means.
- **Implementation*.** Abstract away SW/HW details.

Neural network verification



Vision (v2)

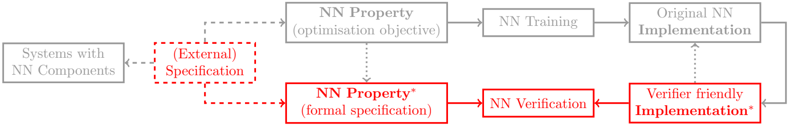
- "Please **check** whether the neural network is safe"

Solution

- **Property*.** Define what "safe" means.
- **Implementation*.** Abstract away SW/HW details.
- **Verification.** Run a state-of-the-art verifier.


SOMETHING IS
MISSING

Existing verification pipeline (1)

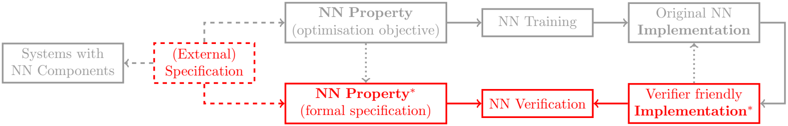


Objective

- Given a neural network $f : \mathbb{R}^n \to \mathbb{R}^m$
- Prove that property $\Xi(f)$ holds

[1] International competition: VNN-COMP (since 2020)

Existing verification pipeline (1)



Objective

- Given a neural network $f : \mathbb{R}^n \to \mathbb{R}^m$
- Prove that property $\Xi(f)$ holds

State of the art

- f is expressed as ONNX or other exchange format
- Tools: Marabou, $\alpha\beta$-CROWN, PyRAT, NNV, ERAN...
- accept $\Xi(f)$ as **linear** pre- and post-conditions on f
- usually expressed in VNN-LIB format[1]

[1] International competition: VNN-COMP (since 2020)

Existing verification pipeline (2)

Established specification language

- Given a neural network $f : \mathbb{R}^n \to \mathbb{R}^m$ in ONNX
- Prove that property $\Xi(f)$, written in VNN-LIB, holds

Existing verification pipeline (2)

Established specification language

- Given a neural network $f : \mathbb{R}^n \to \mathbb{R}^m$ in ONNX
- Prove that property $\Xi(f)$, written in VNN-LIB, holds

Challenges

- Cannot write properties on two networks f_1, f_2
- Properties must list all I/O entries explicitly
- No formal semantics and syntax for VNN-LIB/ONNX
- Cannot automatically bind to dataset entries
- No probabilistic properties

Existing verification pipeline (2)

Established specification language

- Given a neural network $f : \mathbb{R}^n \to \mathbb{R}^m$ in ONNX
- Prove that property $\Xi(f)$, written in VNN-LIB, holds

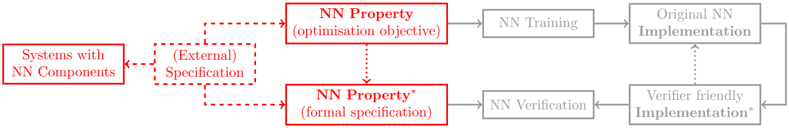
Challenges

- Cannot write properties on two networks f_1, f_2
- Properties must list all I/O entries explicitly
- No formal semantics and syntax for VNN-LIB/ONNX
- Cannot automatically bind to dataset entries
- No probabilistic properties

Existing solutions

- CAISAR: WhyML-like specification language
- Vehicle: dependent-typed functional language

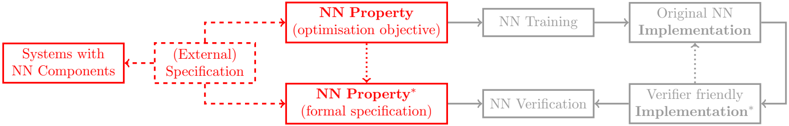
The embedding gap (1)



Objective

- We want to prove a **system** property $\Psi(s(\cdot))$
- But the system s includes a neural network f with (un-) embedding functions $u \circ f \circ e$

The embedding gap (1)



Objective

- We want to prove a **system** property $\Psi(s(\cdot))$
- But the system s includes a neural network f with (un-) embedding functions $u \circ f \circ e$

Strategy

- We decompose it as
  $\Xi(f) \implies \Phi(u \circ f \circ e) \implies \Psi(s(u \circ f \circ e))$
- where
  $\Xi(f)$ is a property of the neural network alone
  $\Phi(u \circ f \circ e)$ maps it back to the problem space

The embedding gap (2)

Objective

- We want to prove a **system** property Ψ
- We decompose it as
  $\Xi(f) \implies \Phi(u \circ f \circ e) \implies \Psi(s(u \circ f \circ e))$

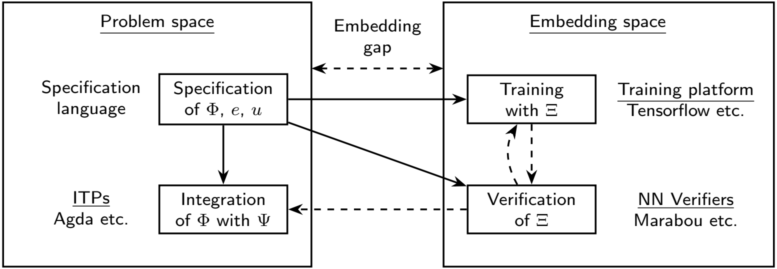
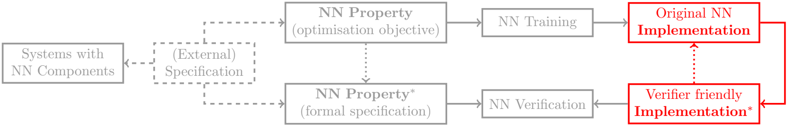


Figure: Outline of Vehicle compiler backends.

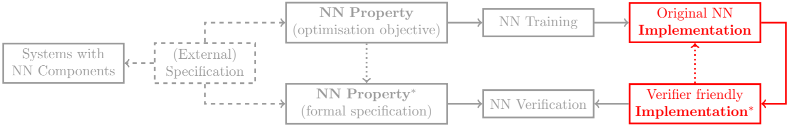
The implementation gap (1)



Mismatch in numerical types

- Original implementation → **finite** precision
- Verified implementation → **real-valued** arithmetic

The implementation gap (1)



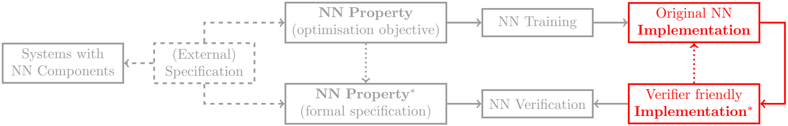
Mismatch in numerical types

- Original implementation $\rightarrow$ **finite** precision
- Verified implementation $\rightarrow$ **real-valued** arithmetic

Floating-point arithmetic

- Real-valued certificates **do not hold** anymore!

The implementation gap (1)



Mismatch in numerical types

- Original implementation → **finite** precision
- Verified implementation → **real-valued** arithmetic

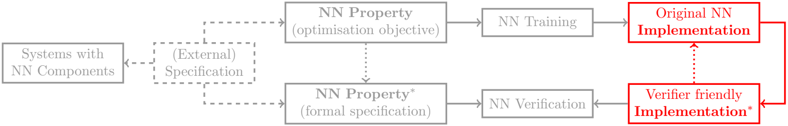
Floating-point arithmetic

- Real-valued certificates **do not hold** anymore!

Quantised neural networks

- Require reasoning about **integer** arithmetic

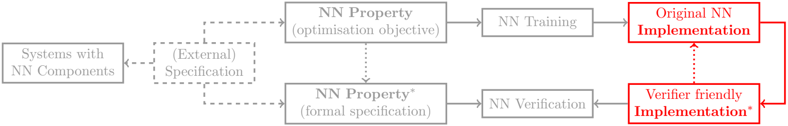
The implementation gap (2)



Many sources of non determinism

- Non-associativity of floating point → e.g. additions

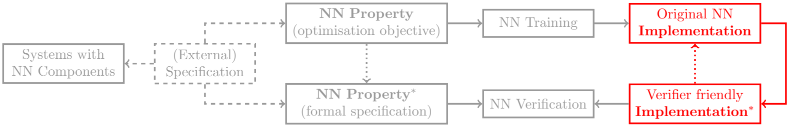
The implementation gap (2)



Many sources of non determinism

- Non-associativity of floating point $\rightarrow$ e.g. additions
- Parallel execution $\rightarrow$ e.g. SIMD, GPUs

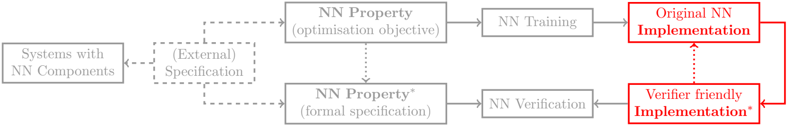
The implementation gap (2)



Many sources of non determinism

- Non-associativity of floating point $\rightarrow$ e.g. additions
- Parallel execution $\rightarrow$ e.g. SIMD, GPUs
- Auto-selection of primitives $\rightarrow$ e.g. convolution algorithm

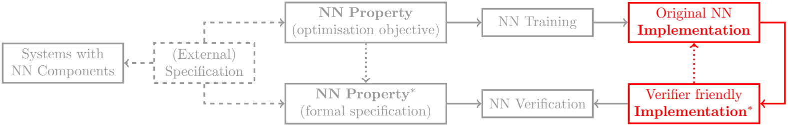
The implementation gap (2)



Many sources of non determinism

- Non-associativity of floating point → e.g. additions
- Parallel execution → e.g. SIMD, GPUs
- Auto-selection of primitives → e.g. convolution algorithm
- Runtime optimisations → e.g. fused layers

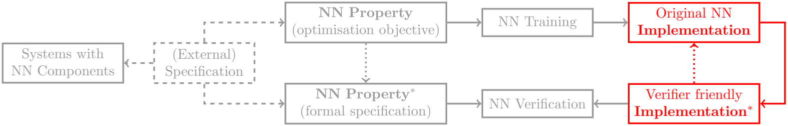
The implementation gap (2)



Many sources of non determinism

- Non-associativity of floating point $\rightarrow$ e.g. additions
- Parallel execution $\rightarrow$ e.g. SIMD, GPUs
- Auto-selection of primitives $\rightarrow$ e.g. convolution algorithm
- Runtime optimisations $\rightarrow$ e.g. fused layers
- Mathematical libraries $\rightarrow$ e.g. rounding of $TanH(x)$

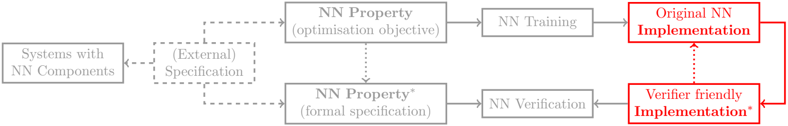
The implementation gap (2)



Many sources of non determinism

- Non-associativity of floating point → e.g. additions
- Parallel execution → e.g. SIMD, GPUs
- Auto-selection of primitives → e.g. convolution algorithm
- Runtime optimisations → e.g. fused layers
- Mathematical libraries → e.g. rounding of $TanH(x)$
- Low-level implementation details → e.g. overflow bugs

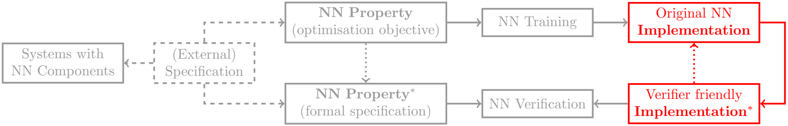
The implementation gap (2)



Many sources of non determinism

- Non-associativity of floating point → e.g. additions
- Parallel execution → e.g. SIMD, GPUs
- Auto-selection of primitives → e.g. convolution algorithm
- Runtime optimisations → e.g. fused layers
- Mathematical libraries → e.g. rounding of $TanH(x)$
- Low-level implementation details → e.g. overflow bugs
- Training → e.g. initialisation, dropout layers

The implementation gap (2)

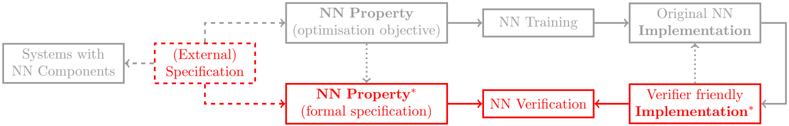


Many sources of non determinism

- Non-associativity of floating point → e.g. additions
- Parallel execution → e.g. SIMD, GPUs
- Auto-selection of primitives → e.g. convolution algorithm
- Runtime optimisations → e.g. fused layers
- Mathematical libraries → e.g. rounding of $TanH(x)$
- Low-level implementation details → e.g. overflow bugs
- Training → e.g. initialisation, dropout layers

Access to software implementation is required!

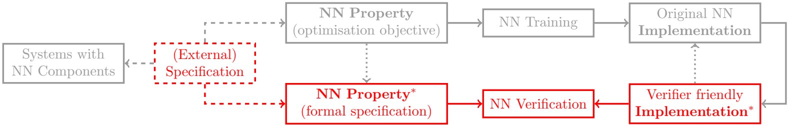
Proof production and checking



ONNX operator support

- Challenge: tool authors need to manually add operators
- E.g. abstract interpretation procedure for new activations
- Solution: great potential for automation here

Proof production and checking



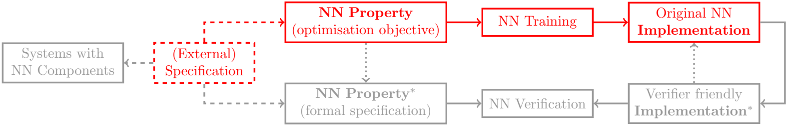
ONNX operator support

- Challenge: tool authors need to manually add operators
- E.g. abstract interpretation procedure for new activations
- Solution: great potential for automation here

Safety proof certificates

- Safety proofs are more difficult to find than counterexamples
- Format: usually some form of branch-and-bound tree
- Challenge 1: formalise under Farkas' lemma
- Challenge 2: write verified proof checkers

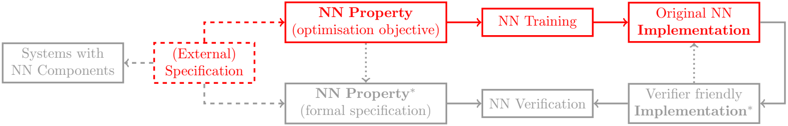
Property-guided training



ML-like approaches

- Data augmentation: (generalised) adversarial training
- Differentiable logic: property $\rightarrow$ loss function

Property-guided training



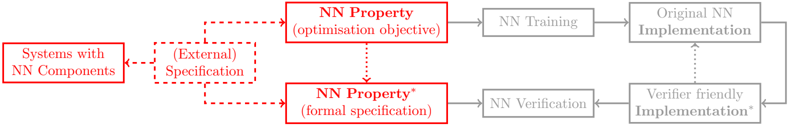
ML-like approaches

- Data augmentation: (generalised) adversarial training
- Differentiable logic: property $\rightarrow$ loss function

Certified approaches

- Interval Bound Propagation (IBP) for robustness training
- Lipschitz-bounded, monotonic & convex neural networks
- Certified training on a limited number of other properties

Verification of cyber-physical systems



CPS

- Feedback controller for plant model (ODEs)
- Requires hybrid description (discrete + continuous)
- International competition: AINNCS category @ ARCH-COMP
- Tools: CORA, JuliaReach, NNV, OVERT, POLAR...
- Challenges: property specification, scalability, software...

Neural network verification today

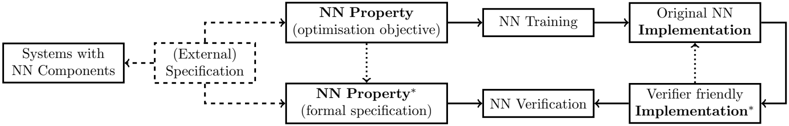
Existing Solutions	High-level		Low-level		Quantised		Software		Future	
PL Challenges	Vehicle	CAISAR	$\alpha\beta$-CROWN	Marabou	QEBVerif	Aster	CBMC	ESBMC	Unified Language	Formal Interfaces
Rigorous Semantics	✓	✓					✓	✓	✓	✓
Embedding Gap	✓								✓	✓
Implementation Gap		✓		✓	✓	✓	✓	✓	✓	
Proof Certificates				✓			✓*	✓*	✓	✓
Supports Training	✓								✓	✓

Five desirable PL features

- Current tools → **partial** support
- We need a more principled solution


THE FUTURE

The future roadmap (1)



Option 1: a unified language

- **Rigorous semantics.** Use dependent types. All components are implemented in a signle language.
- **Embedding gap.** Becomes a type conversion problem.
- **Implementation gap.** Enforce explicit types. No external libraries, unless formally verified.
- **Proof certificates.** The type checker is the proof checker.
- **Training support.** Requires re-implementation or code synthesis in the new language.

The future roadmap (2)

Option 2: formal interfaces

- More **flexible**: can accommodate existing frameworks.
- More **scalable**: avoids complex type checking

The future roadmap (2)

Option 2: formal interfaces

- More **flexible**: can accommodate existing frameworks.
- More **scalable**: avoids complex type checking

Modular design

- Keep a maximally-expressive specification language.
- Design a compiler that can use existing tools.
- Each tool solves a specific verification/synthesis sub-goal.
- Requires formal interfaces and proof certificates.

The future roadmap (2)

Option 2: formal interfaces

- More **flexible**: can accommodate existing frameworks.
- More **scalable**: avoids complex type checking

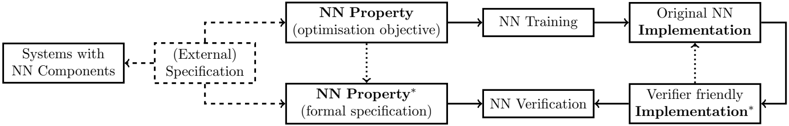
Modular design

- Keep a maximally-expressive specification language.
- Design a compiler that can use existing tools.
- Each tool solves a specific verification/synthesis sub-goal.
- Requires formal interfaces and proof certificates.

Our inspiration

- Behavioural interface specification languages (BISL) such as JML, Why3, SPARK

A diagrammatic summary



Existing Solutions	High-level		Low-level		Quantised		Software		Future	
PL Challenges	Vehicle	CAISAR	$\alpha\beta$-CROWN	Marabou	QEBVerif	Aster	CBMC	ESBMC	Unified Language	Formal Interfaces
Rigorous Semantics	✓	✓					✓	✓	✓	✓
Embedding Gap	✓								✓	✓
Implementation Gap		✓		✓	✓	✓	✓	✓	✓	
Proof Certificates				✓			✓*	✓*	✓	✓
Supports Training	✓								✓	✓

Any questions?