



## Neural Network Verification with Vehicle

Today's presentors: Ekaterina Komendantskaya and Luca Arnaboldi (live),  
Matthew Daggitt (online), on behalf of the Vehicle team

# The Vehicle Team



Matthew Daggitt



Natalia Slusarz



Wen Kokke



Bob Atkey



Luca Arnaboldi



Katya K



Michael Duggan



# Table of Contents



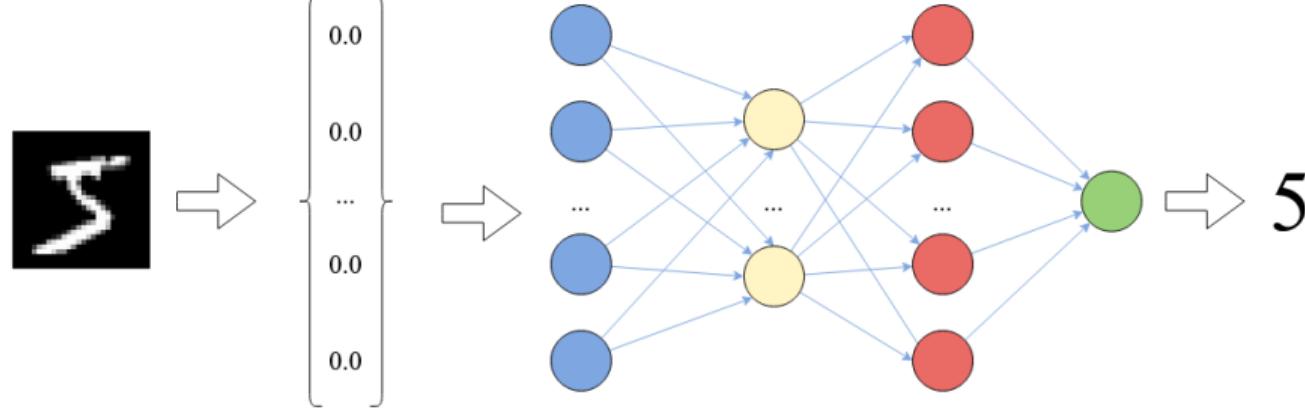
Neural Network Verification: overview of the new domain

The lifecycle of neural network verification

Challenges and Languages

Vehicle's Role and Purpose of this Tutorial

# Neural nets for classification



Formally,

a neural network is a function  $N : R^n \rightarrow R^m$ .

# Neural networks



... are ideal for “perception” tasks:

- ▶ approximate functions when exact solution is hard to get
- ▶ tolerant to noisy and incomplete data

# Neural networks



... are ideal for “perception” tasks:

- ▶ approximate functions when exact solution is hard to get
- ▶ tolerant to noisy and incomplete data

# Neural networks



... are ideal for “perception” tasks:

- ▶ approximate functions when exact solution is hard to get
- ▶ tolerant to noisy and incomplete data

# Neural networks



... are ideal for “perception” tasks:

- ▶ approximate functions when exact solution is hard to get
- ▶ tolerant to noisy and incomplete data

BUT

- ▶ solutions not easily conceptualised (**lack of explainability**)
- ▶ prone to a new range of safety and security problems:

# Neural networks



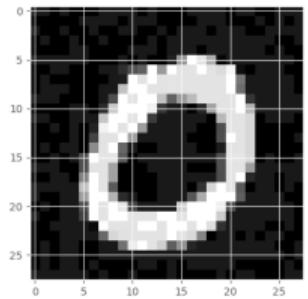
... are ideal for “perception” tasks:

- ▶ approximate functions when exact solution is hard to get
- ▶ tolerant to noisy and incomplete data

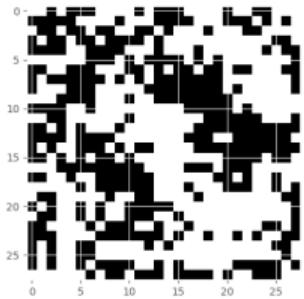
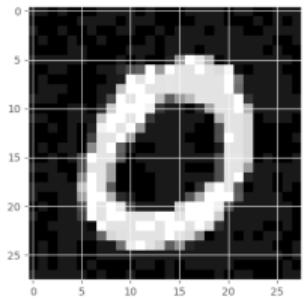
BUT

- ▶ solutions not easily conceptualised (**lack of explainability**)
- ▶ prone to a new range of safety and security problems:
  - adversarial attacks
  - data poisoning
  - catastrophic forgetting

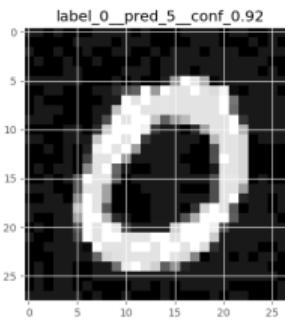
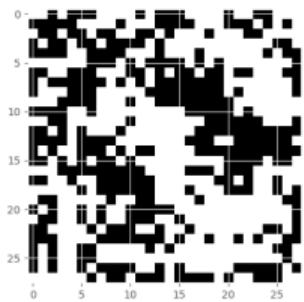
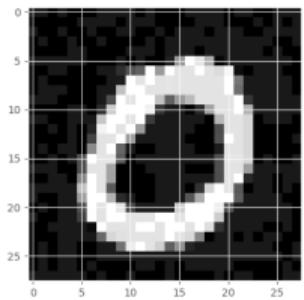
# One example: Adversarial Attacks



# One example: Adversarial Attacks

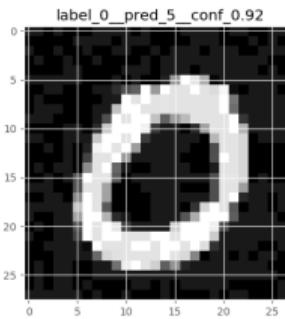
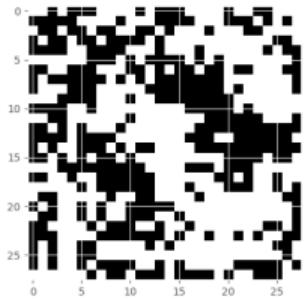
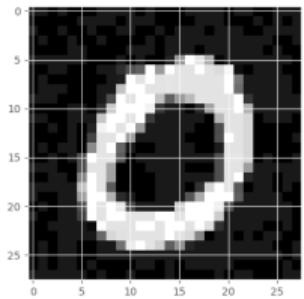


# One example: Adversarial Attacks





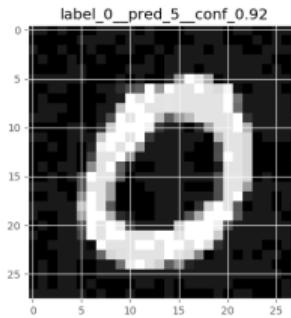
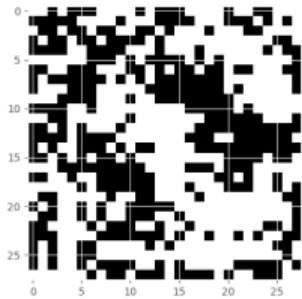
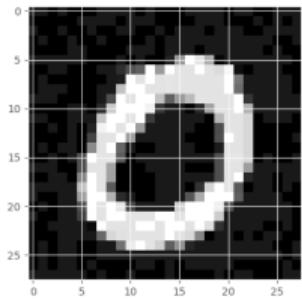
# One example: Adversarial Attacks



the perturbations are imperceptible to human eye



# One example: Adversarial Attacks

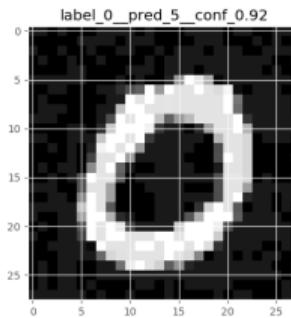
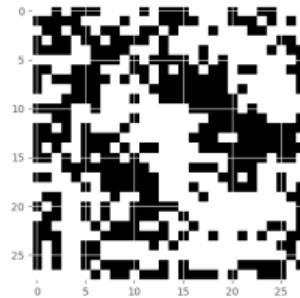
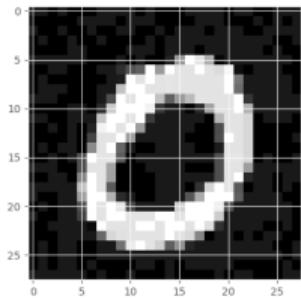


the perturbations are imperceptible to human eye

attacks transfer from one neural network to another

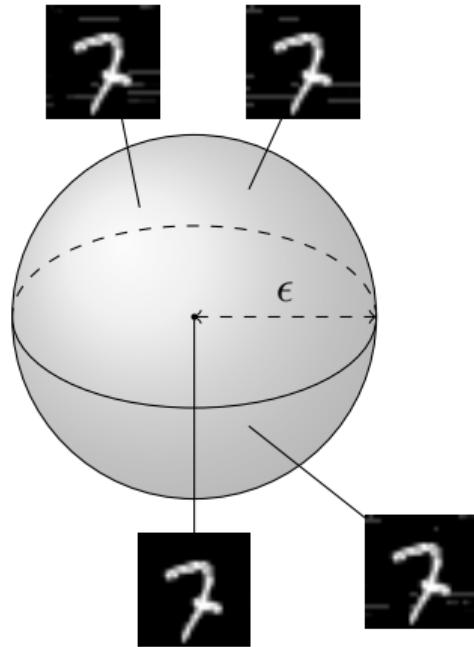


# One example: Adversarial Attacks



the perturbations are imperceptible to human eye  
attacks transfer from one neural network to another  
affect any domain where neural networks are applied

## Verification Property: “ $\epsilon$ -ball robustness”



An  $\epsilon$ -ball  $\mathbb{B}(\hat{\mathbf{x}}, \epsilon) = \{\mathbf{x} \in \mathbb{R}^n : |\hat{\mathbf{x}} - \mathbf{x}| \leq \epsilon\}$

Classify all points in  $\mathbb{B}(\hat{\mathbf{x}}, \epsilon)$  “robustly”.



## Another example property: ACAS Xu

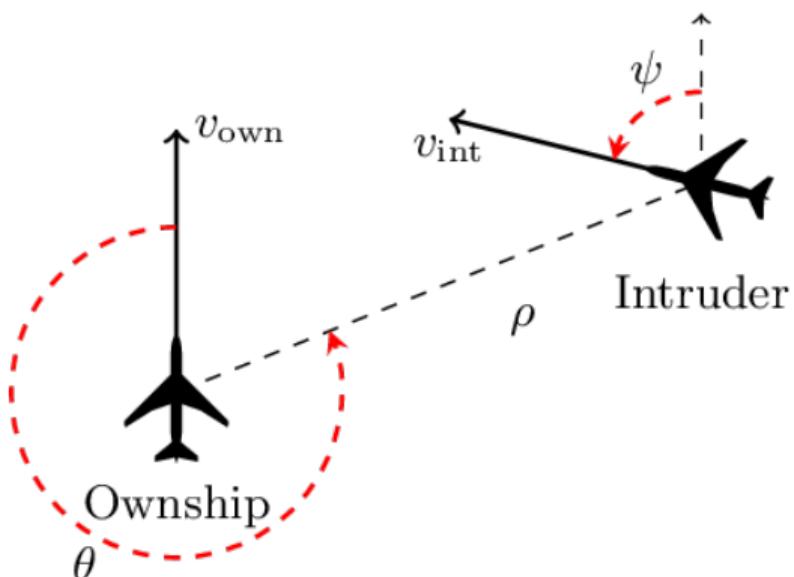
A collision avoidance system for unmanned autonomous aircraft.

Inputs:

- ▶ Distance to intruder,  $\rho$
- ▶ Angle to intruder,  $\theta$
- ▶ Intruder heading,  $\varphi$
- ▶ Speed,  $v_{own}$
- ▶ Intruder speed,  $v_{int}$

Outputs:

- ▶ Clear of conflict
- ▶ Strong left
- ▶ Weak left
- ▶ Weak right
- ▶ Strong right





The system was originally implemented as a 2Gb lookup table but was replaced with a neural network in order to improve size and latency requirements.



The system was originally implemented as a 2Gb lookup table but was replaced with a neural network in order to improve size and latency requirements.

**10 different specified properties in total.**



The system was originally implemented as a 2Gb lookup table but was replaced with a neural network in order to improve size and latency requirements.

**10 different specified properties in total.**

## Definition (ACAS Xu: Property 1)

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.



The system was originally implemented as a 2Gb lookup table but was replaced with a neural network in order to improve size and latency requirements.

10 different specified properties in total.

#### Definition (ACAS Xu: Property 1)

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.

$$\begin{aligned} & (\rho \geq 55947.691) \wedge (v_{own} \geq 1145) \wedge (v_{int} \leq 60) \\ \Rightarrow & \text{the score for COC is at most 1500} \end{aligned}$$

## More Generally



Given  $N : R^n \rightarrow R^m$

Verification of such functions most commonly boils down to specifying admissible intervals for the function's output given an interval for its inputs.

## More Generally

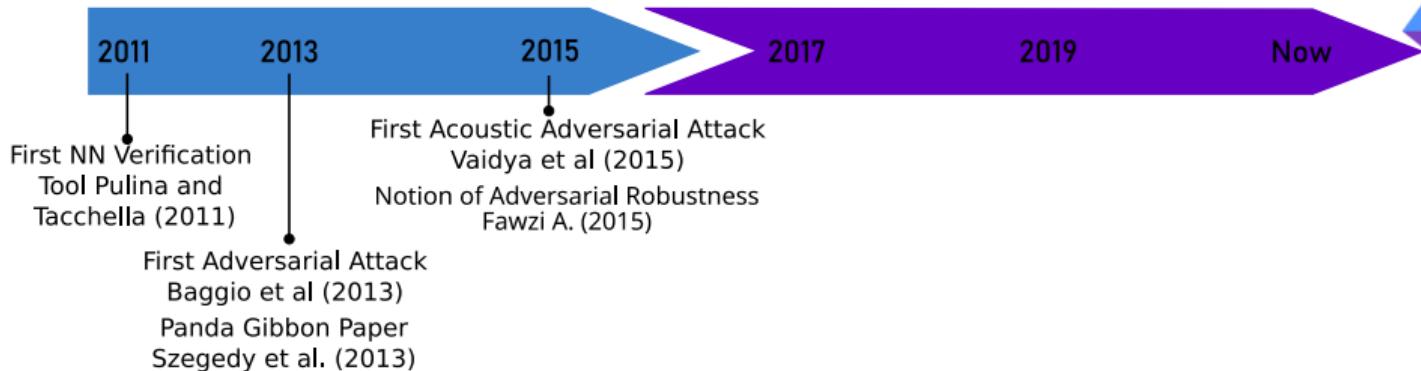


Given  $N : R^n \rightarrow R^m$

Verification of such functions most commonly boils down to specifying admissible intervals for the function's output given an interval for its inputs.

-  Casadio, M., Komendantskaya, E., Daggitt, M.L., Kokke, W., Katz, G., Amir, G., Refaeli, I.: Neural network robustness as a verification property: A principled case study. In: Computer Aided Verification (CAV 2022).

# Overview of The Verification Landscape



I have this specification  
I want to verify!



property specification

What tools are available?  
**2015**



approximate

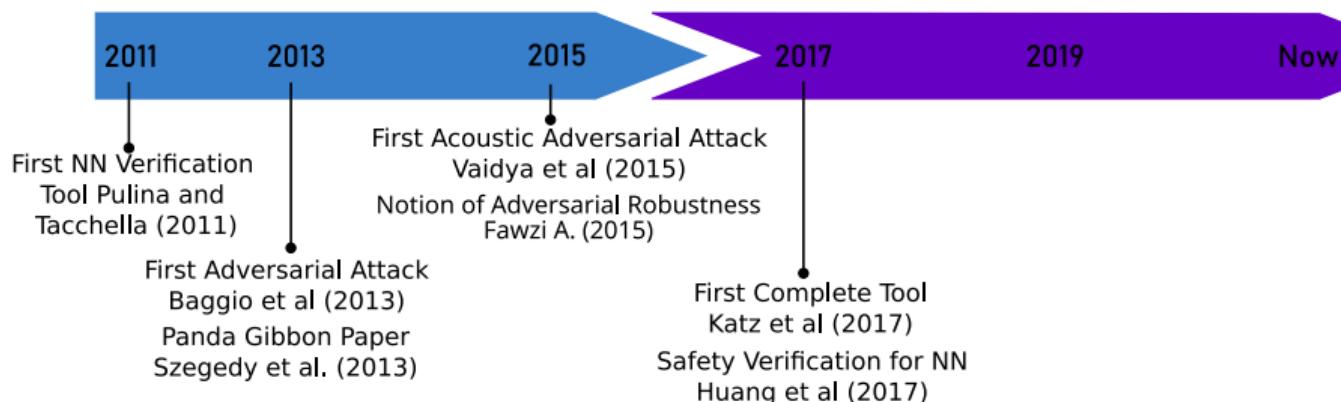


adversarial

complete

others

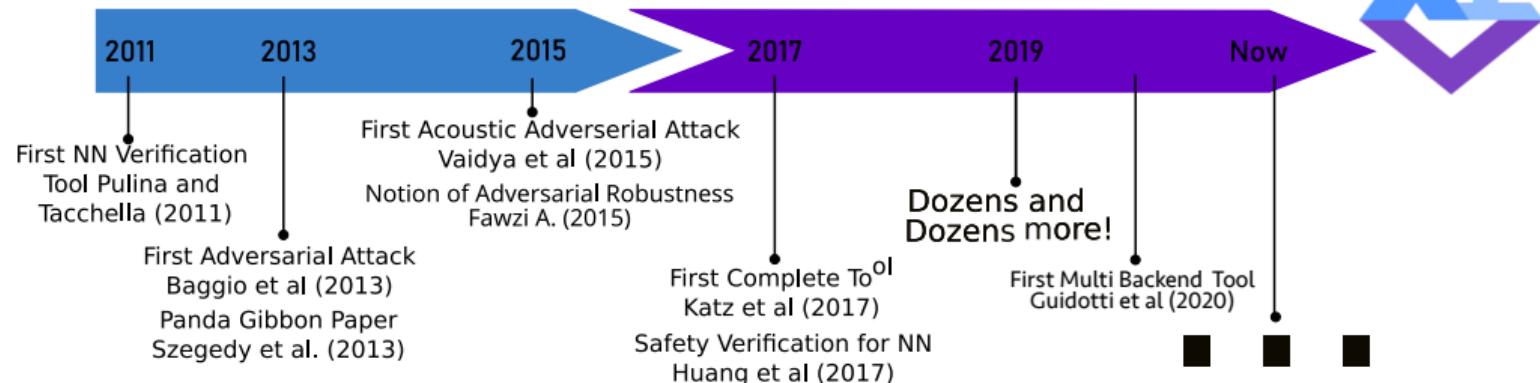
# Overview of The Verification Landscape



What tools are available?  
**2017**



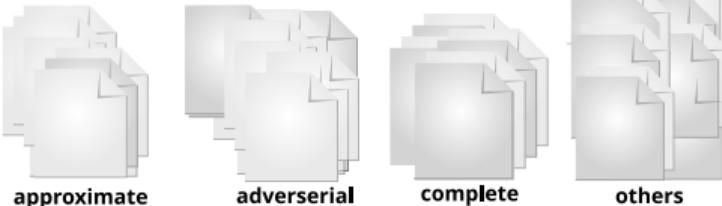
# Overview of The Verification Landscape



I have this specification  
I want to verify!



What tools are available?  
**2023**



## Current Verifier Landscape



A whole range of domain-specific verifiers exist:



## Current Verifier Landscape

A whole range of domain-specific verifiers exist:

- ▶ Marabou (SMT technology)



## Current Verifier Landscape

A whole range of domain-specific verifiers exist:

- ▶ Marabou (SMT technology)
- ▶ ERAN (abstract interpretation + MILP)



# Current Verifier Landscape

A whole range of domain-specific verifiers exist:

- ▶ Marabou (SMT technology)
- ▶ ERAN (abstract interpretation + MILP)
- ▶ Verisig (interval arithmetic)
- ▶ AlphaBetaCROWN (linear bound propagation)
- ▶ ...

## International Standards and Competitions

<https://www.vnnlib.org/>



# Current Verifier Landscape

A whole range of domain-specific verifiers exist:

- ▶ Marabou (SMT technology)
- ▶ ERAN (abstract interpretation + MILP)
- ▶ Verisig (interval arithmetic)
- ▶ AlphaBetaCROWN (linear bound propagation)
- ▶ ...

## International Standards and Competitions

<https://www.vnnlib.org/>

Marabou is our current choice as it is complete, and the set of expressible queries is large!

-  Guy Katz, Clarke Barrett, D. Dill, K. Julian, and M. Kochenderfer. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In CAV, 2017.

# Table of Contents



Neural Network Verification: overview of the new domain

The lifecycle of neural network verification

Challenges and Languages

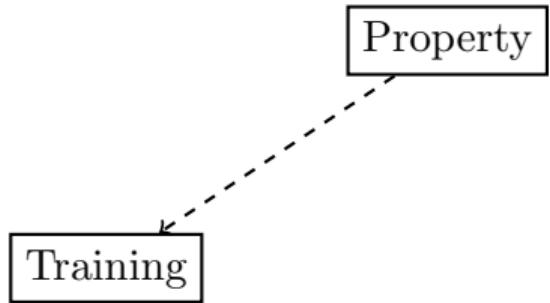
Vehicle's Role and Purpose of this Tutorial

# The lifecycle of neural network verification



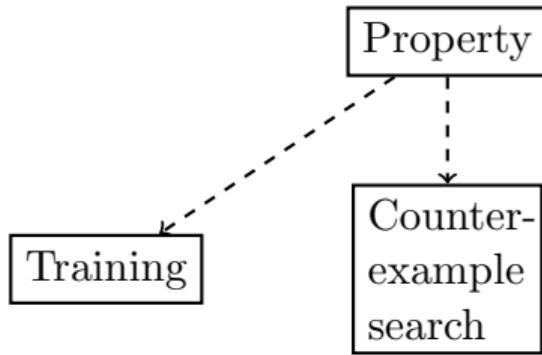
Property

# The lifecycle of neural network verification



DL2  
ACT  
etc.

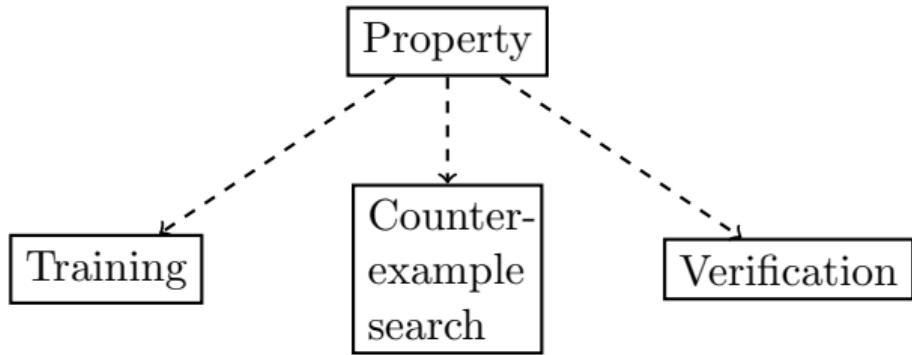
# The lifecycle of neural network verification



DL2  
ACT  
etc.

PGD  
FGSM  
etc.

# The lifecycle of neural network verification



DL2

ACT

etc.

PGD

FGSM

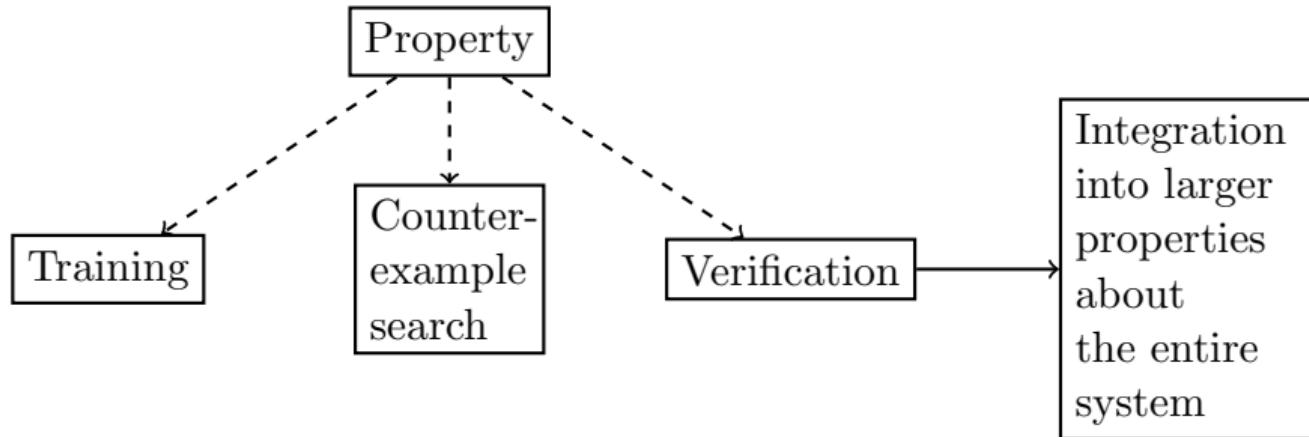
etc.

Marabou

Eran

etc.

# The lifecycle of neural network verification

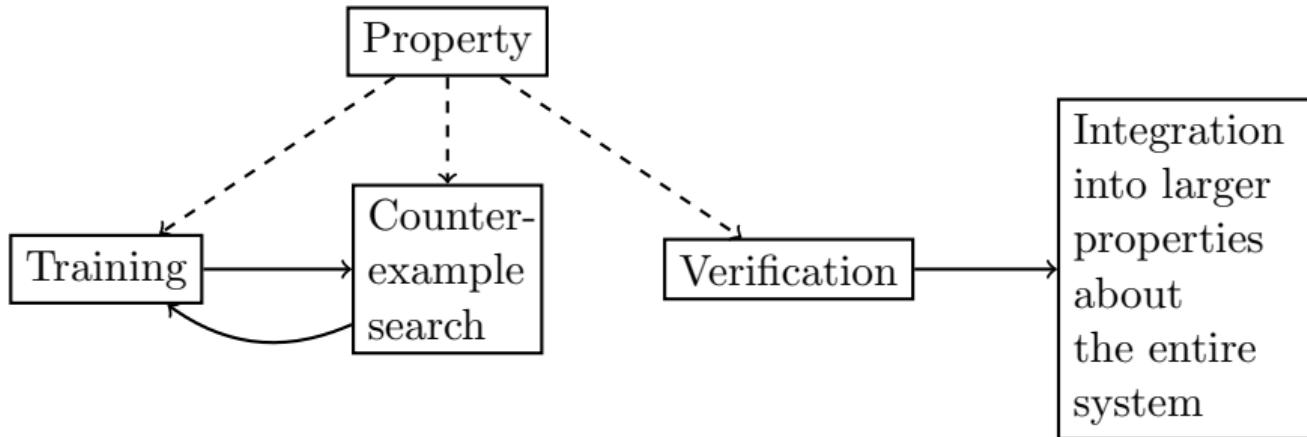


DL2  
ACT  
etc.

PGD  
FGSM  
etc.

Marabou  
Eran  
etc.

# The lifecycle of neural network verification

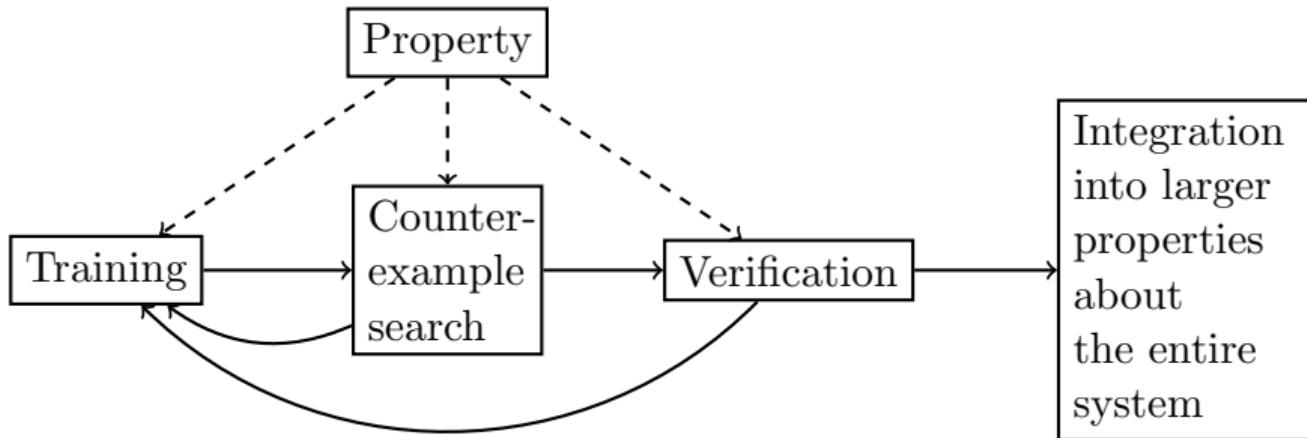


DL2  
ACT  
etc.

PGD  
FGSM  
etc.

Marabou  
Eran  
etc.

# The lifecycle of neural network verification

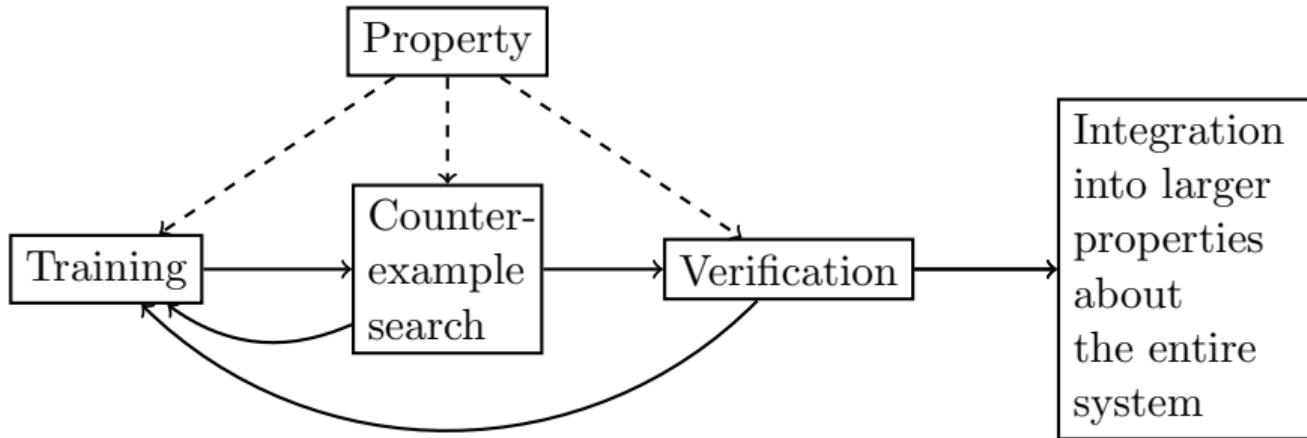


DL2  
ACT  
etc.

PGD  
FGSM  
etc.

Marabou  
Eran  
etc.

# The lifecycle of neural network verification



DL2  
ACT  
etc.

PGD  
FGSM  
etc.

Marabou  
Eran  
etc.

# Table of Contents



Neural Network Verification: overview of the new domain

The lifecycle of neural network verification

Challenges and Languages

Vehicle's Role and Purpose of this Tutorial



## Challenges the area faces

- ▶ Theory: finding appropriate verification properties

## Challenges the area faces



- ▶ Theory: finding appropriate verification properties
- ▶ Solvers: undecidability of non-linear real arithmetic and scalability of neural network verifiers

## Challenges the area faces



- ▶ Theory: finding appropriate verification properties
- ▶ Solvers: undecidability of non-linear real arithmetic and scalability of neural network verifiers
- ▶ ML: understanding and integrating property-driven training

## Challenges the area faces



- ▶ Theory: finding appropriate verification properties
- ▶ Solvers: undecidability of non-linear real arithmetic and scalability of neural network verifiers
- ▶ ML: understanding and integrating property-driven training
- ▶ Programming: finding the right languages to support these developments

## Challenges the area faces



- ▶ Theory: finding appropriate verification properties
- ▶ Solvers: undecidability of non-linear real arithmetic and scalability of neural network verifiers
- ▶ ML: understanding and integrating property-driven training
- ▶ Programming: finding the right languages to support these developments
- ▶ Complex systems: integration of neural net verification into complex systems



Some of these problems are aggravated by insufficient programming language or API support

Lets look under the hood...



## Training framework: DL2

```
126     class RobustnessConstraint(Constraint):
127
141         def get_domains(self, x_batches, y_batches):
142             assert len(x_batches) == 1
143             n_batch = x_batches[0].size()[0]
144
145             return [[Box(np.clip(x_batches[0][i].cpu().numpy() - self.eps, 0, 1),
146                         np.clip(x_batches[0][i].cpu().numpy() + self.eps, 0, 1))
147                         for i in range(n_batch)]]
148
149         def get_condition(self, z_inp, z_out, x_batches, y_batches):
150             n_batch = x_batches[0].size()[0]
151             z_out = transform_network_output(z_out, self.network_output)[0]
152             #z_logits = F.log_softmax(z_out[0], dim=1)
153
154             pred = z_out[np.arange(n_batch), y_batches[0]]
155
156             limit = torch.FloatTensor([0.3])
157             if self.use_cuda:
```

# Training framework: ART



```
333     @classmethod
334     def property6a(cls, dom: AbsDom):
335         p = AcasProp(name='property6a', dom=dom, safe_fn='cols_is_min', viol_fn='cols_not_min',
336                       fn_args=[AcasOut.CLEAR_OF_CONFLICT])
337         p.set_input_bound(AcasIn.RHO, new_low=12000, new_high=62000)
338         p.set_input_bound(AcasIn.THETA, new_low=0.7, new_high=3.141592)
339         p.set_input_bound(AcasIn.PSI, new_low=-3.141592, new_high=-3.141592 + 0.005)
340         p.set_input_bound(AcasIn.V_OWN, new_low=100, new_high=1200)
341         p.set_input_bound(AcasIn.V_INT, new_low=0, new_high=1200)
342         p.set_all_applicable_as(False)
343         p.set_applicable(1, 1, True)
344         return p
```

-  Lin, X., Zhu, H., Samanta, R., and Jagannathan, S. (2020). Art: Abstraction refinement-guided training for provably correct neural networks. In FMCAD 2020

# Verification framework: Marabou



```
def test_acas_1_1_normalize():
    """
    Test the 1,1 experimental ACAS Xu network.
    By passing "normalize=true" to read_nnet, Marabou adjusts the parameters of the first and last layers of the
    network to incorporate the normalization.
    As a result, properties can be defined in the original input/output spaces without any manual normalization.
    """
    filename = "acasxu/ACASXU_experimental_v2a_1_1.nnet"
    testInputs = [
        [1000.0, 0.0, -1.5, 100.0, 100.0],
        [10000.0, -3.0, -1.5, 300.0, 300.0],
        [5000.0, -3.0, 0.0, 300.0, 600.0]
    ]
    testOutputs = [
        [177.87553729, 173.75796115, 193.05920806, 153.07876146, 195.00495022],
        [-0.55188079, 0.46863711, 0.44250383, 0.44151988, 0.43959133],
        [29.9190734, 27.2386958, 45.02497222, 14.5610455, 46.86448056]
    ]
    network = evaluateFile(filename, testInputs, testOutputs, normalize = True)
```



Katz, G., Huang, D. A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljic, A., Dill, D. L., Kochenderfer, M. J., and Barrett, C. W. (2019). The Marabou framework for verification and analysis of deep neural networks. In CAV 2019

# Verification framework: ERAN



```
1 [12000, 62000]
2 [0.7, 3.141592][-3.141592, -0.7]
3 [-3.141592, -3.136592]
4 [100, 1200]
5 [0, 600]
```

---

```
1 5
2 y0 min
```



Singh, G., Gehr, T., Püschel, M., and Vechev, M. T. (2019). An abstract domain for certifying neural networks. *PACMPL*, 3(POPL):41:1–41:30.



## Verification property language: VNNLIB

```
28  (assert (or
29      (and (<= X_0 0.700434925) (>= X_0 -0.129289109)
30          (<= X_1 0.499999896) (>= X_1 0.11140846)
31          (<= X_2 -0.499204121) |(>= X_2 -0.499999896)
32          (<= X_3 0.5) (>= X_3 -0.5)
33          (<= X_4 0.5) (>= X_4 -0.5))
34      (and (<= X_0 0.700434925) (>= X_0 -0.129289109)
35          (<= X_1 -0.11140846) (>= X_1 -0.499999896)
36          (<= X_2 -0.499204121) (>= X_2 -0.499999896)
37          (<= X_3 0.5) (>= X_3 -0.5)
38          (<= X_4 0.5) (>= X_4 -0.5)))
39 )
40
41 ; unsafe if coc is not minimal
42 (assert (or
43     (and (<= Y_1 Y_0))
44     (and (<= Y_2 Y_0)))
45     (and (<= Y_3 Y_0)))
46     (and (<= Y_4 Y_0)))
47 ))
48
```

Recap: What are the problems from the PL perspective?



Recap: What are the problems from the PL perspective?



- I<sup>O</sup>* Interoperability – properties are not portable between training/counter-example search/ verification.

## Recap: What are the problems from the PL perspective?



- $I^O$  Interoperability – properties are not portable between training/counter-example search/ verification.
- $I^P$  Interpretability – code is not easy to understand.

## Recap: What are the problems from the PL perspective?



- $I^O$  Interoperability – properties are not portable between training/counter-example search/ verification.
- $I^P$  Interpretability – code is not easy to understand.
- $I^J$  Integration – properties of networks cannot be linked to larger control system properties.

## Recap: What are the problems from the PL perspective?



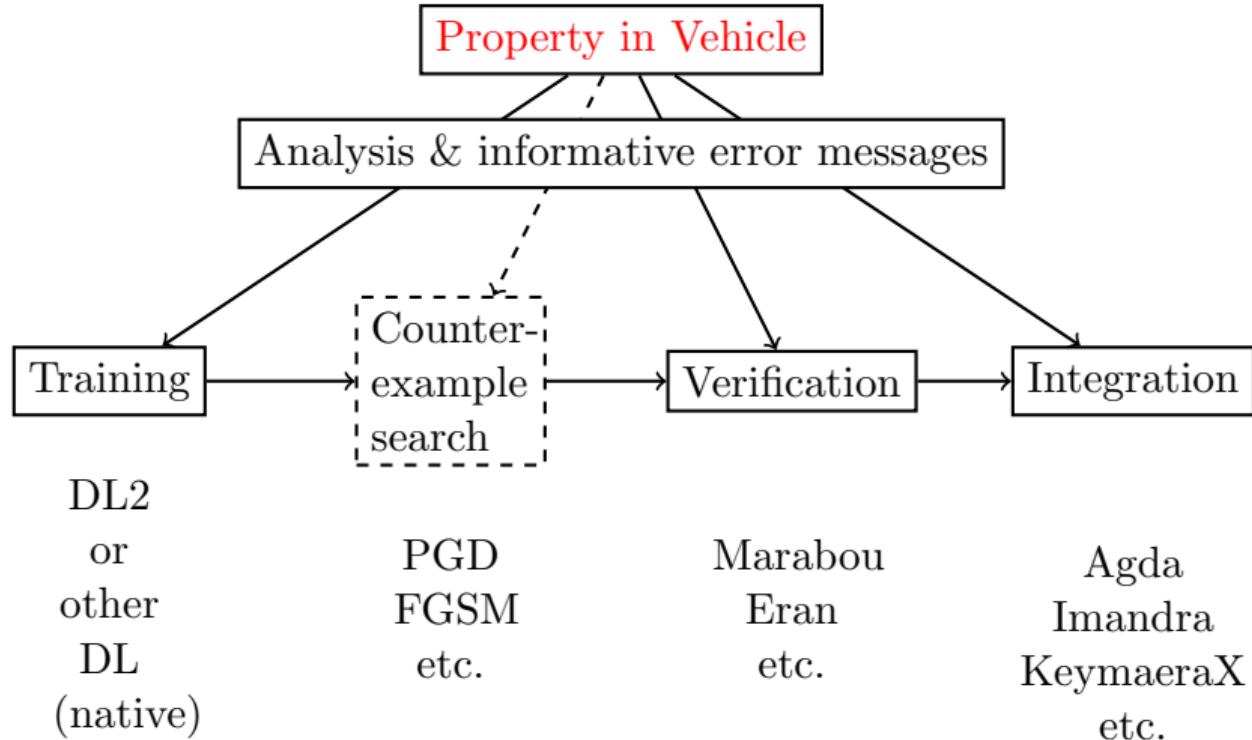
- $I^O$  Interoperability – properties are not portable between training/counter-example search/ verification.
- $I^P$  Interpretability – code is not easy to understand.
- $I^I$  Integration – properties of networks cannot be linked to larger control system properties.
- $E^G$  Embedding gap – little support for translation between problem space (as in original spec) and input space (at neural network level).

Vehicle is designed to address all of these problems

# Vehicle ...



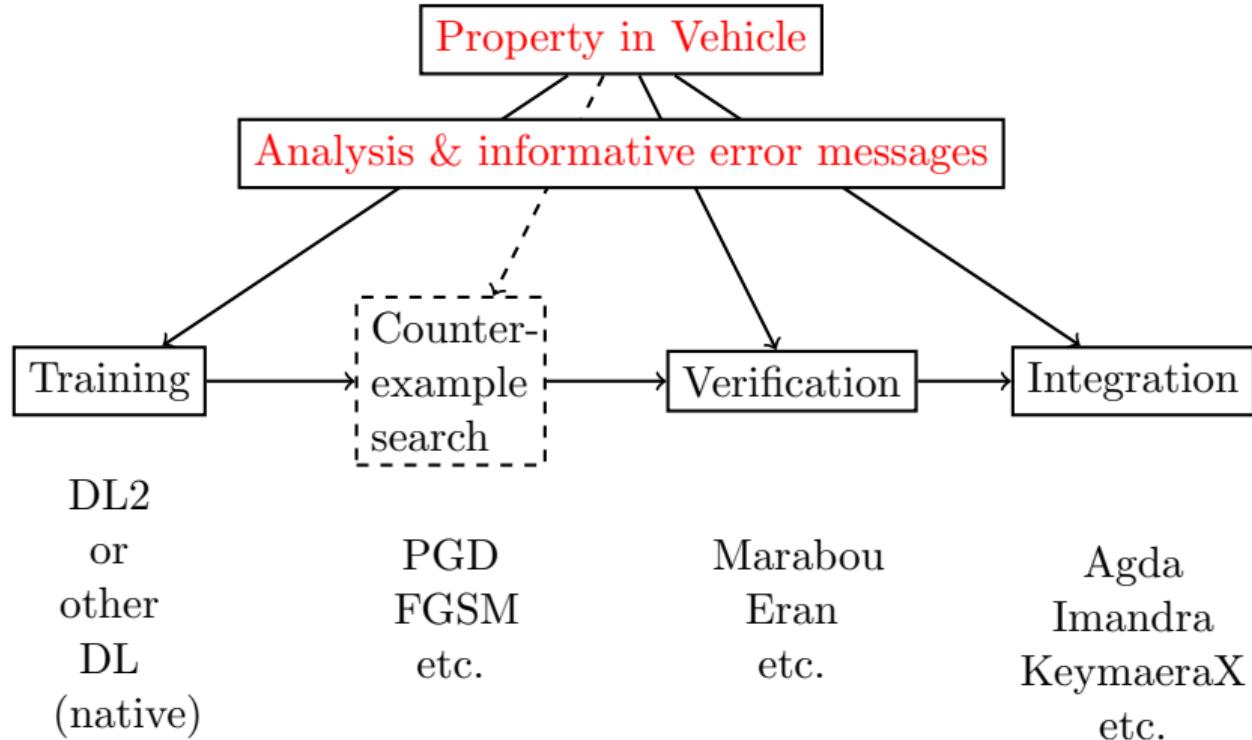
is a domain-specific functional language for writing high-level property specifications for neural networks



# Vehicle ...



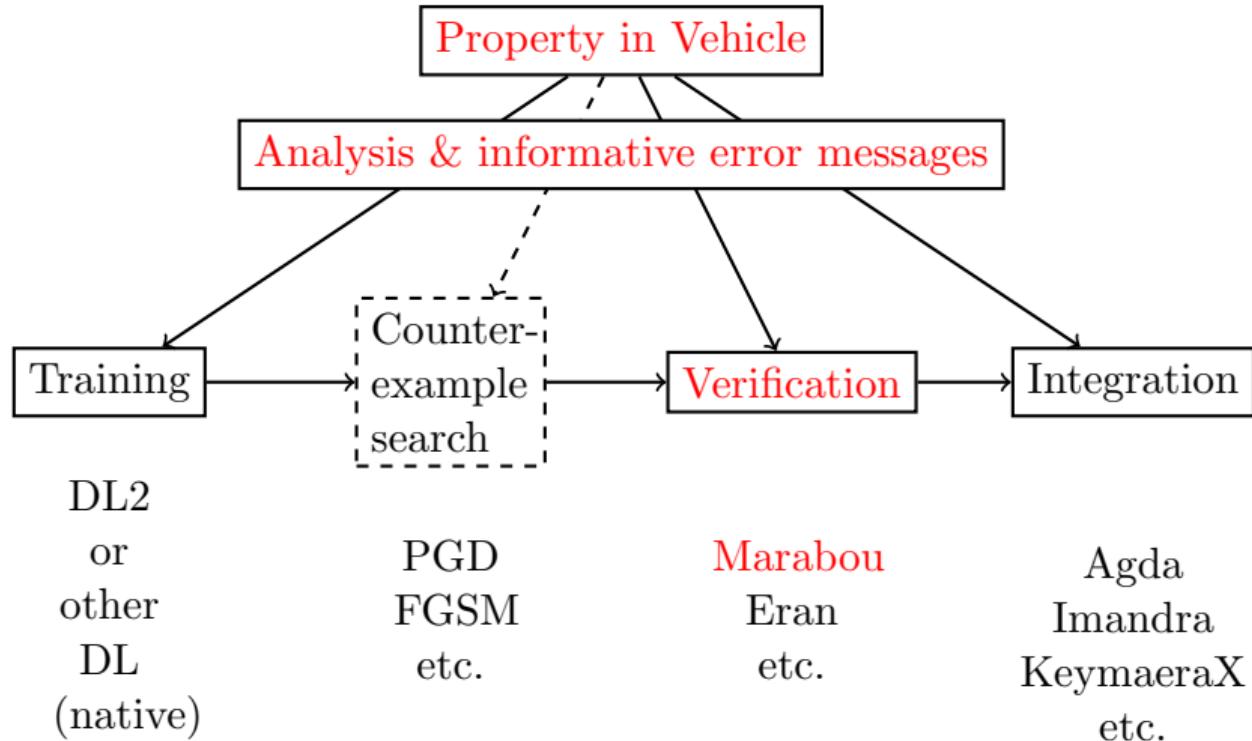
is a domain-specific functional language for writing high-level property specifications for neural networks





# Vehicle ...

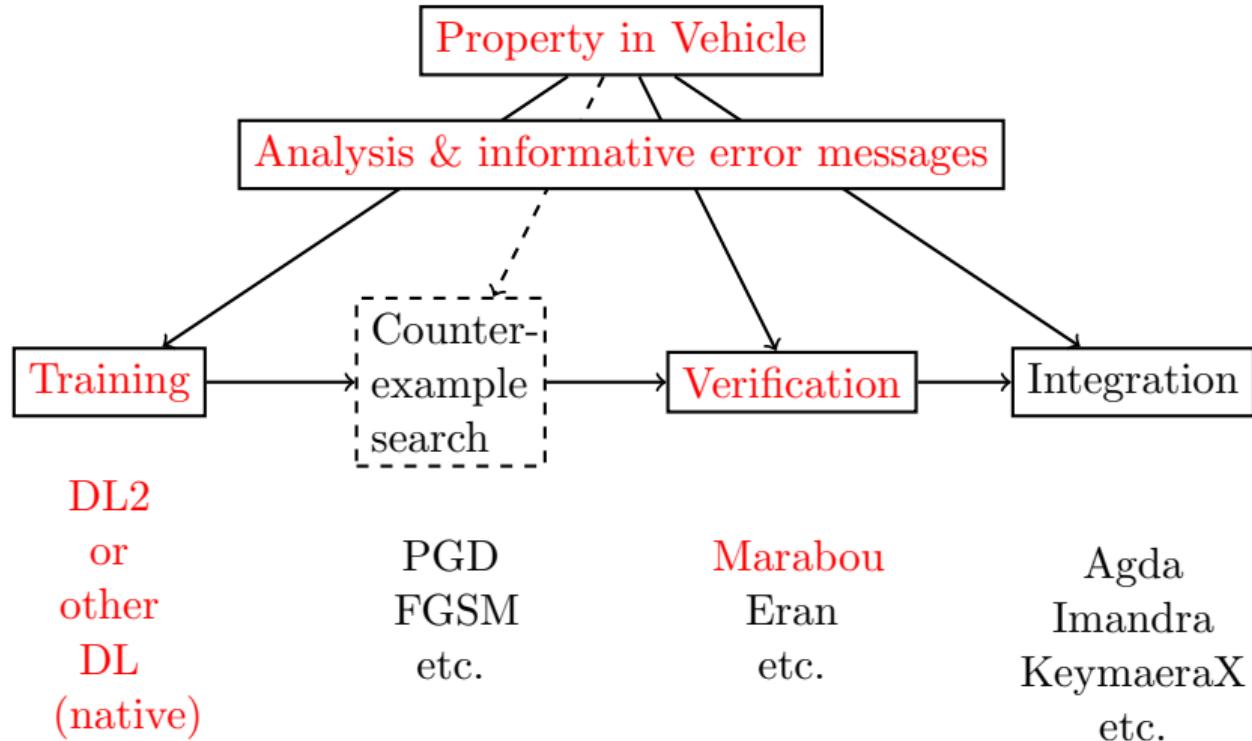
is a domain-specific functional language for writing high-level property specifications for neural networks





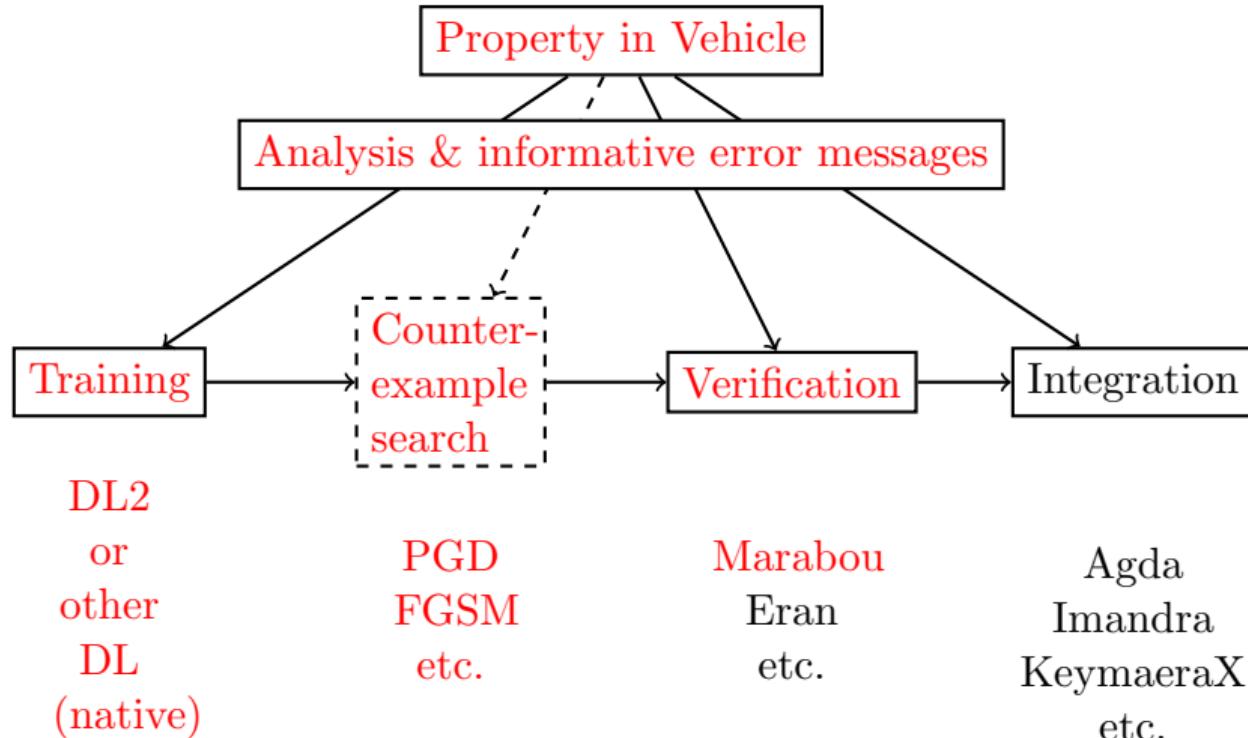
# Vehicle ...

is a domain-specific functional language for writing high-level property specifications for neural networks



# Vehicle ...

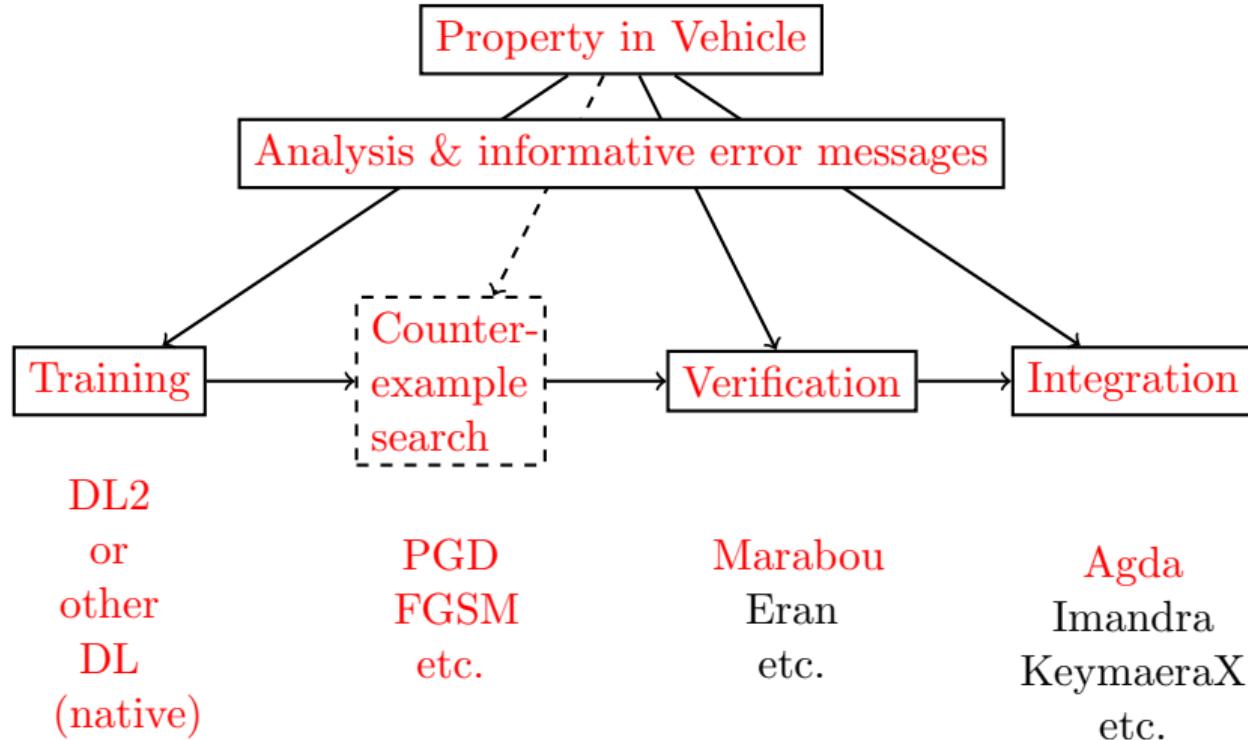
is a domain-specific functional language for writing high-level property specifications for neural networks



# Vehicle ...



is a domain-specific functional language for writing high-level property specifications for neural networks





## Other Similar APIs

- ▶ Socrates [in Python]: Given a spec and a network (in JSON), calls different NN verifiers.



L. H. Pham, J. Li, and J. Sun. 2020. SOCRATES: Towards a Unified Platform for Neural Network Verification. CoRR abs/2007.11206.

Left unresolved:  $I^O, I^P, I^f, E^G$



## Other Similar APIs

- ▶ Socrates [in Python]: Given a spec and a network (in JSON), calls different NN verifiers.
  - ❑ L. H. Pham, J. Li, and J. Sun. 2020. SOCRATES: Towards a Unified Platform for Neural Network Verification. CoRR abs/2007.11206.
- Left unresolved:  $I^O, I^P, I^f, E^G$
- ▶ NeVer 2.0 [in Python]: added training, pruning and quantization to this functionality.
  - ❑ D. Guidotti, L. Pulina, and A. Tacchella. 2020. NeVer 2.0: Learning, Verification and Repair of Deep Neural Networks. CoRR abs/2011.09933.
- Resolved:  $I^O$  (partially). Left untesolved:  $I^P, I^f, E^G$



## Other Similar APIs

- ▶ Socrates [in Python]: Given a spec and a network (in JSON), calls different NN verifiers.
  - ❑ L. H. Pham, J. Li, and J. Sun. 2020. SOCRATES: Towards a Unified Platform for Neural Network Verification. CoRR abs/2007.11206.  
Left unresolved:  $I^O, I^P, I^f, E^G$
- ▶ NeVer 2.0 [in Python]: added training, pruning and quantization to this functionality.
  - ❑ D. Guidotti, L. Pulina, and A. Tacchella. 2020. NeVer 2.0: Learning, Verification and Repair of Deep Neural Networks. CoRR abs/2011.09933.  
Resolved:  $I^O$  (partially). Left untesolved:  $I^P, I^f, E^G$
- ▶ CoCoNet [in Python]: NN format converter with GUI
  - ❑ D. Guidotti, A. Tacchella, L. Pulina, S. Demarchi 2023. <http://neuralverification.org/>  
Resolved:  $I^P$  (partially). Left untesolved:  $I^O, I^f, E^G$



## Other Similar APIs

- ▶ Socrates [in Python]: Given a spec and a network (in JSON), calls different NN verifiers.
  - ❑ L. H. Pham, J. Li, and J. Sun. 2020. SOCRATES: Towards a Unified Platform for Neural Network Verification. CoRR abs/2007.11206.  
Left unresolved:  $I^O, I^P, I^f, E^G$
- ▶ NeVer 2.0 [in Python]: added training, pruning and quantization to this functionality.
  - ❑ D. Guidotti, L. Pulina, and A. Tacchella. 2020. NeVer 2.0: Learning, Verification and Repair of Deep Neural Networks. CoRR abs/2011.09933.  
Resolved:  $I^O$  (partially). Left untesolved:  $I^P, I^f, E^G$
- ▶ CoCoNet [in Python]: NN format converter with GUI
  - ❑ D. Guidotti, A. Tacchella, L. Pulina, S. Demarchi 2023. <http://neuralverification.org/>  
Resolved:  $I^P$  (partially). Left untesolved:  $I^O, I^f, E^G$
- ▶ Caisar [in OCAML] – general specification language and connection to several NN Verifiers
  - ❑ J. Girard-Satabin, M. Alberti, F. Bobot, Z. Chihani, and A. Lemesle. 2022. CAISAR: A platform for Characterizing Artificial Intelligence Safety and Robustness. In AISafety.  
Resolved:  $I^P, I^f$ . Left unresolved:  $I^O, E^G$

# Table of Contents



Neural Network Verification: overview of the new domain

The lifecycle of neural network verification

Challenges and Languages

Vehicle's Role and Purpose of this Tutorial



## Vehicle's Aim...

... is to resolve the problems  $I^O$ ,  $I^P$ ,  $I^f$ ,  $E^G$

# Vehicle's Aim...



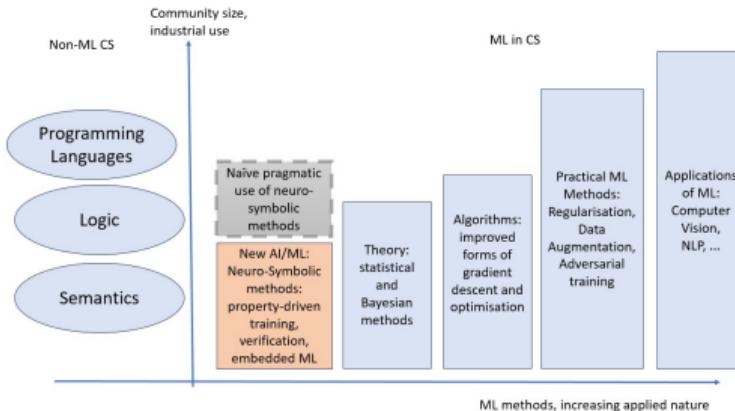
... is to resolve the problems  $I^O$ ,  $I^P$ ,  $I^f$ ,  $E^G$

... and support community's effort towards resolution of the “Grand Challenges”

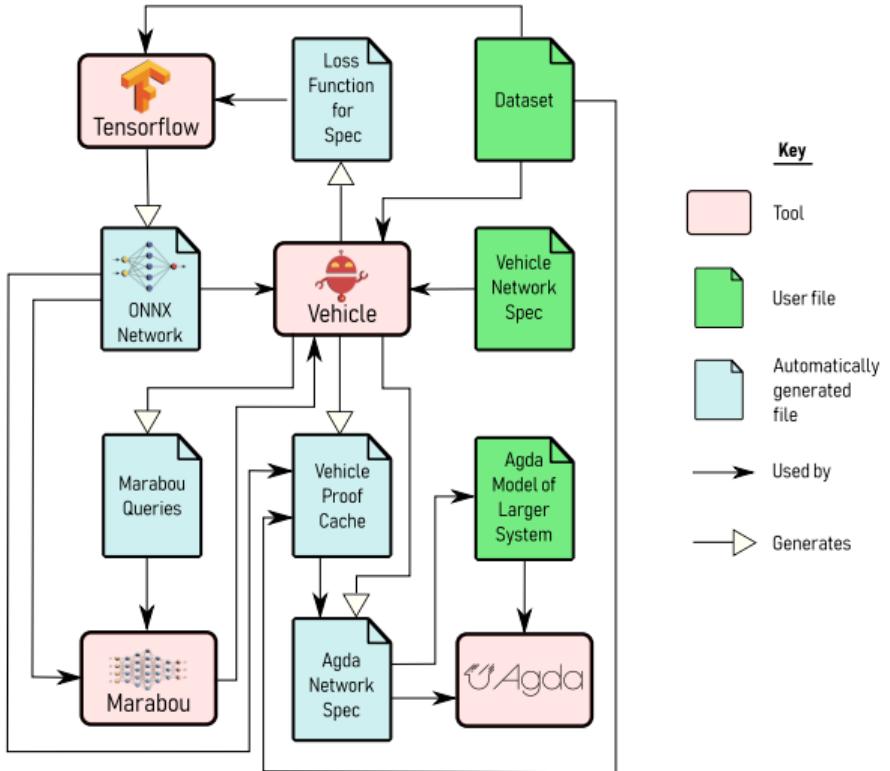
# “Grand Challenges” & Vehicle



- ▶ Theory: finding appropriate verification properties
- ▶ Solvers: undecidability of non-linear real arithmetic and scalability of neural network verifiers
- ▶ ML: understanding and integrating property-driven training
- ▶ Programming: finding the right languages to support these developments
- ▶ Complex systems: integration of neural net verification into complex systems



# Vehicle Architecture



# Sources



-  M. Daggitt, R. Atkey, W. Kokke, E. Komendantskaya, L. Arnaboldi: Compiling Higher-Order Specifications to SMT Solvers: How to Deal with Rejection Constructively. CPP 2023
-  N. Slusarz, E. Komendantskaya, M. Daggitt, R. Stewart, K. Stark: Logic of Differentiable Logics: Towards a Uniform Semantics of DL. LPAR 2023.
-  Matthew L. Daggitt, Wen Kokke, Robert Atkey, Luca Arnaboldi, Ekaterina Komendantskaya: Vehicle: Interfacing Neural Network Verifiers with Interactive Theorem Provers. FOMLAS 2022
-  Vehicle Team: The Vehicle language: <https://github.com/vehicle-lang> 2023.
-  M.Daggitt and W.Kokke: Vehicle User Manual. <https://vehicle-lang.readthedocs.io> 2023.
-  The Vehicle team: Vehicle Tutorial <https://vehicle-lang.github.io>. 2023. Tutorial code repository <https://github.com/vehicle-lang/vehicle-tutorial>.

# Purpose of this Tutorial...



- ▶ Introduce **Vehicle** specification language at the user level
- ▶ Discuss challenges in NN verification ("grand" and technical)
- ▶ Gather feedback

# Plan for the rest of this tutorial



- ▶ Before coffee break:
  - ▶ Brief introduction to **Vehicle** specification language
  - ▶ Neural Network Robustness: an iconic verification case
- ▶ During and after the break:
  - ▶ **Exercise session:** write and verify a **Vehicle** spec
    - Tutorial pages: <https://vehicle-lang.github.io>
    - Join tutorial Slack channel via the tutorial page, to ask questions
- ▶ After the break:
  - ▶ Property-driven training: theory and demo in Vehicle
  - ▶ Application areas for NN Verification and large system integration