

Specification is All You Need

Ekaterina Komendantskaya^{1,2}[0000–0002–3240–0987],
Marco Casadio²[0009–0001–7675–0743], Jessica Ciupa^{3,2}[0009–0003–9505–0608],
Thomas Flinkow⁴[0000–0002–8075–2194], Joe Hall¹, Colin Kessler^{2,3}, and
Alistair Sirman¹[0009–0009–3342–089X]

¹ University of Southampton, Southampton, UK
{E.Komendantskaya,A.Sirman}@soton.ac.uk

² Heriot-Watt University, Edinburgh, UK
{mc248,ck2049}@hw.ac.uk

³ University of Edinburgh, Edinburgh, UK

⁴ Maynooth University, Maynooth, Ireland
thomas.flinkow@mu.ie

Abstract. Neural networks are now routinely deployed in cyber-physical systems, yet they frequently fail to learn critical safety and correctness properties from data alone, and so must be formally verified before deployment. Verification, however, only reports unsafe behaviour; it cannot repair the network. Repair requires training methods that integrate logical specifications directly into the optimisation objective. Adversarial training does this for a single property, robustness to small perturbations within ϵ -cubes, whereas cyber-physical and control systems need it for arbitrary safety specifications. Over the past decade a number of approaches to this generalisation have been proposed, and many of them have now been incorporated into Vehicle, a domain-specific language for specifying neural networks. Vehicle offers a readable syntax for specifications and a compiler that type-checks them and translates each specification into loss functions for training in Python, into queries for neural network verifiers, and into definitions for interactive theorem provers. Because one specification drives training, verification and integration alike, Vehicle has been adopted for teaching and research into safe autonomous systems at postgraduate level, notably in a course of the UKRI Centre for Doctoral Training in Dependable and Deployable AI for Robotics in Edinburgh. In this invited paper we distil the experience of several PhD students who were among the first users of specification-driven training in Vehicle, through three case studies: the ACAS Xu collision-avoidance networks, a neural controller for a gliding drone, and neural controllers for drug dosing. Our aim is to persuade the reader to try, and eventually adopt, Vehicle for similar projects across robotics and autonomous systems.

Keywords: Neural Network Verification · Vehicle · Specification-driven Training · Adversarial Training · Neuro-symbolic AI · Differentiable Logics · Formal Methods · Machine Learning

1 Introduction

Neural networks have been shown to frequently fail to satisfy properties of interest after training [36,11], and even the most accurate networks are susceptible to adversarial attacks,

i.e. inputs that resemble the training data but fall outside of the learnt distribution [59,31]. This restricts their use in safety-critical domains. To address this gap, numerous formal verification tools⁵ have been proposed over the past decade, including Marabou [36,37,70], Branch-and-Bound [7], NNV [61,45], and α , β -CROWN [75,72,73,67,74,55], the winner of the most recent Neural Network Verification Competition (VNN-COMP) [38].

Verification, however, only reports whether a trained network satisfies a property. It cannot repair a network that fails the verification, and standard data-driven training rarely produces networks that satisfy non-trivial properties. Verification therefore has to be combined with training methods that take the property into account. This gives rise to a *specification-driven training* cycle: a property is specified; a network is trained towards it; the trained network is verified; and the failures reported by the verifier feed back into training (Fig. 1).

Adversarial training. In the simplest case, adversarial training [31,46] improves a network’s robustness: for each training input $\vec{x}$ it finds, by the Fast Gradient Sign Method (FGSM) or Projected Gradient Descent (PGD), the worst-case perturbation $\vec{x}'$ within an ϵ -cube around $\vec{x}$ and trains on it (Section 2.2). The property it trains for is *local robustness* of the network f , $\forall \vec{x}. \|\vec{x} - \vec{x}'\| \leq \epsilon \longrightarrow \|f(\vec{x}) - f(\vec{x}')\| \leq \delta$: the prediction may move by at most δ when the input moves by at most ϵ .

Although these methods generally work well for computer vision models, they do not generalise beyond this domain. In many scenarios we encounter properties of the form $\forall \vec{x}. \mathcal{P}(\vec{x}) \longrightarrow Q(f(\vec{x}))$, where preconditions $\mathcal{P}$ and postconditions Q differ substantially from local robustness. First, $\mathcal{P}$ and Q do not have to relate to specific data points: global properties constrain the input space directly. Secondly, $\mathcal{P}$ and Q do not have to be atomic, and can be given by arbitrary logical formulae.

Global properties and non-atomic postconditions. Neural networks deployed in cyber-physical systems as *neural controllers* are the most common source of such properties. Their inputs are sensor readings with a physical meaning, such as velocity, distance and angle, so safe and unsafe input regions come as bounds on these readings. A user writes the precondition $\mathcal{P}$ as any Boolean combination of such bounds, and the compiler extracts from it a *hyper-rectangle* (Section 2.1), the generalisation of the ϵ -cube of adversarial training [9,10,27]. The seminal neural network verification benchmark ACAS Xu⁶, for example, is defined by ten such properties, and many neural controllers yield similar verification conditions [48]. Their postconditions are rarely atomic either [54,21]: a controller is not prescribed an action in every scenario, only forbidden the unsafe ones. In ACAS Xu, the precondition “the distance to the intruder is dangerously small” does not mandate “advise ‘strong left’” but “do not advise ‘clear of conflict’”, that is, the disjunction “advise ‘weak left’, ‘strong left’, ‘weak right’, or ‘strong right’”. *Differentiable logics* [22,64,26] (Section 2.2) translate such formulae into loss terms, generalising the robustness constraint that FGSM and PGD train for to a loss that encodes an arbitrary postcondition $Q(f(\vec{x}))$.

⁵ For an overview of the state-of-the art in formal verification of neural networks, we refer the interested reader to [34,63,44,4].

⁶ The experimental neural network compression [35] of the unmanned variant ACAS Xu of the Airborne Collision Avoidance System X (ACAS X) [41,40].

Vehicle. Over the past years these ideas have been consolidated in *Vehicle* [16,13,17], a domain-specific language for writing specifications of neural networks. A *Vehicle* specification states the type of a network and the properties it should satisfy. As Fig. 1 shows, the *Vehicle* compiler type-checks the specification and compiles it into a loss function for training in Python and into queries for a neural network verifier and into definitions for interactive theorem provers; the last of these is not used in this paper, and we refer the reader to [56] for a case study that builds on it. One artefact thus drives every stage of the cycle, which is the thesis of our title: given the specification, training and verification follow from it. For the same reason *Vehicle* has been adopted for teaching and research at postgraduate level, notably in a course of the UKRI Centre for Doctoral Training in Dependable and Deployable AI for Robotics⁷ in Edinburgh and in the Cyber-Physical Systems group at Southampton, where students carried out the case studies of this paper.

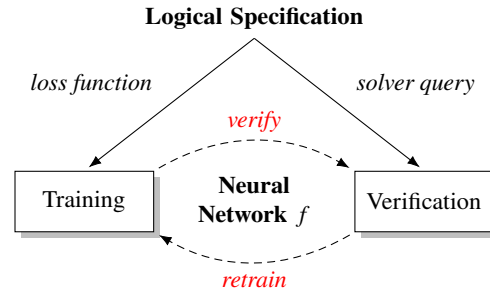


Fig. 1: A diagrammatic depiction of the specification-driven training [17] enabled by *Vehicle*.

Contributions. This invited paper reports on that experience.

- We recall the optimisation objective that combines hyper-rectangles with differentiable logics, for properties $\forall \vec{x}. \mathcal{P}(\vec{x}) \longrightarrow Q(f(\vec{x}))$ in which $\mathcal{P}$ and Q are quantifier-free first-order formulae, the network f occurs in Q but not in $\mathcal{P}$, and $\mathcal{P}$ has constant lower and upper bounds in each dimension (Section 3.1); and we show that this objective is exactly what *Vehicle* compiles a specification to (Section 3.2).
- We present three case studies carried out by PhD students who were among the first users of specification-driven training in *Vehicle*: the ACAS Xu properties (Section 4.1); a neural controller for a gliding drone inspired by the seeds of *Alsomitra macrocarpa* [39], which generalises ACAS Xu to regression and, unlike ACAS Xu, comes with released training data (Section 4.2); and neural controllers for drug dosing [56] (Section 4.3).
- We record what worked and what did not: the choice of differentiable logic, the adversarial search inside the compiled loss, the balance between data and property in the objective, and the properties that turned out to be vacuous or in conflict with the data.

Our aim is to persuade the reader to try, and eventually adopt, *Vehicle* for similar projects across robotics and autonomous systems.

⁷ <https://www.cdt-d2air.uk/>

Paper outline. Section 2 sets the context: input region specification in Section 2.1 and differentiable logics in Section 2.2. Section 3 gives the combined optimisation objective (Section 3.1) and shows how a Vehicle specification is compiled to it (Section 3.2). Section 4 presents the three case studies, and Section 5 concludes with related work, limitations and future work.

2 Context

2.1 Input Region Specification

Verification of neural networks has focused primarily on *robustness verification*, which requires a network $f: \mathbb{R}^m \rightarrow \mathbb{R}^n$ to behave the same on every element of each subspace $S_i \subseteq \mathbb{R}^m$: the same class label for a classifier, an output within δ for a regressor. For computer vision tasks, these subspaces take the shape of ϵ -balls around the input vectors in the data set. Formally, given an input $\vec{x}$ and distance function $\|\cdot\|$, the ϵ -ball around $\vec{x}$ of radius ϵ is defined as:

$$\mathbb{B}(\vec{x}; \epsilon) := \{\vec{x}' \in \mathbb{R}^m : \|\vec{x} - \vec{x}'\| \leq \epsilon\} \quad (1)$$

In practice, it is common to use the ℓ_∞ norm [31], in which case these ϵ -balls are also called ϵ -cubes.

The ϵ -bounded perturbations work well for computer vision [68,32], where an image is a vector in a continuous space and every point near a valid image is another valid image, so that an ϵ -cube around a data point is a meaningful region. The inputs of a neural controller are different: the distance to the intruder in ACAS Xu runs over tens of kilometres and its heading over $[-\pi, \pi]$, and the velocities, angles and positions of the Alsomitra drone in Section 4.2 span ranges that differ by two orders of magnitude. A property of such a controller speaks about a region of sensor readings, for example “*the intruder is distant and slower*”, and that region is not an ϵ -cube around any data point. It is a box with a lower and an upper bound in each dimension, and a specification may contain several such boxes, some of them not populated by data at all. Hyper-rectangles of this kind have been used systematically for verification and specification-driven training in [10,9,27].

A multi-dimensional rectangle (or *hyper-rectangle*) is a shape defined by lower and upper bounds for each dimension, i.e. $\vec{l}, \vec{u} \in \mathbb{R}^m$ such that $\vec{l} \leq \vec{u}$ (component-wise). The hyper-rectangle $H(\vec{l}, \vec{u})$ is then defined as

$$H(\vec{l}, \vec{u}) := \{\vec{x} \in \mathbb{R}^m \mid l_i \leq x_i \leq u_i, \text{ for all } 1 \leq i \leq m\}. \quad (2)$$

As visualised in Fig. 2, hyper-rectangles allow the specification of ϵ -cubes, as well as more flexible local regions, and they can also be used to specify global input regions that are not relative to specific inputs; the precondition of the ACAS Xu property in Eq. (8) is one such global hyper-rectangle.

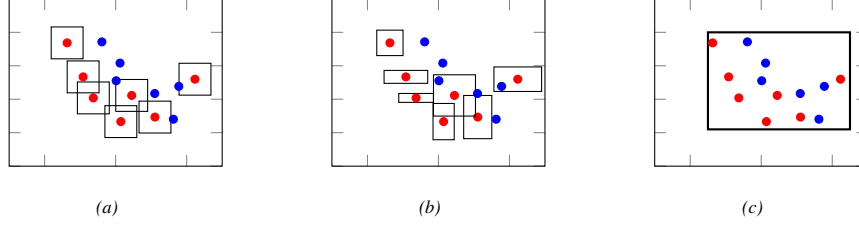


Fig. 2: A visualisation of ϵ -cubes (Fig. 2a), hyper-rectangles relative to each data point (Fig. 2b), and a global hyper-rectangle (Fig. 2c). Red dots represent training data, and blue dots represent test data.

2.2 Specification-driven Training

Training for robustness. In standard machine learning, for a model f_θ parametrised by θ , optimal parameters θ^* are obtained by using gradient descent to minimise a *loss* $\mathcal{L}$ that quantifies the difference between the network’s actual output $f_\theta(\vec{x})$ and the desired output $\vec{y}$, shown in Eq. (3):

$$\theta^* = \arg \min_{\theta} \mathbb{E}_{(\vec{x}, \vec{y}) \sim \mathcal{D}} [\mathcal{L}(\vec{x}, \vec{y}; f_\theta)] \quad (3)$$

Adversarial training via FGSM and PGD algorithms [31,46] first finds adversarial examples and integrates them into the training process. Formally, the minimisation objective changes from the one shown in Eq. (3) to the one shown in Eq. (4), where $\vec{x}'$ is an adversarial example within the ϵ -cube around a given input $\vec{x}$.

$$\theta^* = \arg \min_{\theta} \mathbb{E}_{(\vec{x}, \vec{y}) \sim \mathcal{D}} \left[\max_{\vec{x}' \in \mathbb{B}(\vec{x}; \epsilon)} \mathcal{L}(\vec{x}', \vec{y}; f_\theta) \right] \quad (4)$$

Therefore, adversarial training consists of a two-step process: an inner maximisation problem to select the worst-case adversarial perturbation within the ϵ -cube around a clean data point $\vec{x}$, followed by adapting the network weights to reduce the loss caused by this adversarial perturbation. Madry et al. [46] proposed to use PGD to approximately solve the inner optimisation problem.

Training to satisfy arbitrary logical constraints. Adversarial training improves prediction accuracy on perturbed inputs, that is, it trains for one fixed property: similar inputs should be assigned the same class label. *Differentiable logics* have been proposed to generate additional loss functions from arbitrary logical constraints instead. This additional loss term calculates a penalty depending on how much the network deviates from satisfying the constraint. Popular choices for differentiable logics include both systems designed specifically for use in neural networks, such as *Deep Learning with Differentiable Logics* (DL2) [22], as well as already existing and well-studied logical systems such as *fuzzy logics*. The first in-depth study investigating the use of fuzzy logics as loss functions (called *Differentiable Fuzzy Logics*) was provided by [64] and presented insights into the learning characteristics of differentiable logic operators. A unifying framework of

Table 1: An overview of some popular differentiable logics, and of the additive quantitative linear logic (QLL) used in this paper. $\llbracket \cdot \rrbracket$ denotes the mapping of a logical formula into loss; for QLL, $p \in (0, \infty)$ is the hardness.

Logic	Domain	$\llbracket \top \rrbracket$	$\llbracket \perp \rrbracket$	$\llbracket \neg x \rrbracket$	$\llbracket x \wedge y \rrbracket$	$\llbracket x \vee y \rrbracket$
DL2 [22]	$[0, \infty)$	0	∞	undefined	$x + y$	xy
Gödel logic [64]	$[0, 1]$	1	0	$1 - x$	$\min\{x, y\}$	$\max\{x, y\}$
Product logic [64]	$[0, 1]$	1	0	$1 - x$	xy	$x + y - xy$
QLL [8,25]	$[-\infty, \infty]$	$-\infty$	∞	$-x$	$\frac{1}{p} \log(e^{px} + e^{py})$	$-\frac{1}{p} \log(e^{-px} + e^{-py})$

differentiable logics including DL2, fuzzy logics, and Signal Temporal Logic (STL) [66], called *Logic of Differentiable Logics (LDL)*, is provided in [57] and allows for mechanised proofs of theoretical properties of differentiable logics [1]. Table 1 lists the connectives of DL2, of two fuzzy logics and of the logic used in this paper.

Differentiable logics and optimisation. In the specification-driven training of [22,26], all constraints are of the form $\forall \vec{x}' \in \mathbb{B}(\vec{x}; \epsilon). \phi$, i.e. a logical constraint should not only hold for a specific input $\vec{x}$, but also in its close vicinity. The optimisation objective for the constraint ϕ is then the one shown in Eq. (5):

$$\theta^* = \arg \min_{\theta} \mathbb{E}_{(\vec{x}, \vec{y}) \sim \mathcal{D}} \left[\alpha \mathcal{L}(\vec{x}, \vec{y}; f_{\theta}) + (1 - \alpha) \max_{\vec{x}' \in \mathbb{B}(\vec{x}; \epsilon)} \llbracket \phi \rrbracket(\vec{x}, \vec{x}', \vec{y}; f_{\theta}) \right]. \quad (5)$$

Here, $\mathcal{L}$ denotes the prediction loss⁸, $(\vec{x}, \vec{y}) \sim \mathcal{D}$ is a pair of training data $\vec{x}$ and target $\vec{y}$ (i.e. the true label for classification, or the target value for regression), and α is a hyperparameter required to balance the different loss terms.

The choice of differentiable logic. There are many differentiable logics to choose from, and these differ not only in terms of their domains and operators, but also when it comes to satisfying properties such as soundness, compositionality, smoothness, and shadow-lifting (which requires the truth value of a conjunction to increase whenever the truth value of one of its conjuncts increases). It has been shown [1,66] that no differentiable logic can satisfy three desirable properties simultaneously: shadow-lifting, idempotence and associativity. However, Capucci et al. [8] showed that the logical operations can be “softened”, or parametrised, so that shadow-lifting and associativity hold exactly and idempotence holds in a soft form. Soft idempotence converges to standard idempotence as the parameter p goes to infinity, and the parametrised connectives converge to the min and max operators of Gödel logic. Such a logic is well suited both to optimisation, at finite p , and to a Boolean reading, in the limit. We therefore use it for all experiments in this paper, and refer the reader to [25] for a detailed comparison of this logic with other differentiable logics.

⁸ Such as cross-entropy loss $\mathcal{L}_{\text{CE}}$ for classification, mean-squared-error loss $\mathcal{L}_{\text{MSE}}$ for regression, etc.

Quantitative linear logic. The logic is the additive fragment of *quantitative linear logic* (QLL) [8,25], a family of logics indexed by a hardness $p \in (0, \infty)$ with both a Boolean and a quantitative reading. At $p = \infty$ the logic is multiplicative additive linear logic, whose additive connectives are interpreted by max and min; for finite p its quantitative interpretation is a differentiable loss, and [8] proves that this loss is adequate for the Boolean interpretation, so that a formula is satisfied exactly when its loss is minimal. In the additive interpretation, formulae take values in $[-\infty, \infty]$, and smaller is truer: $\llbracket \top \rrbracket = -\infty$, $\llbracket \perp \rrbracket = \infty$, and a positive value measures the margin by which a formula is violated. Atoms are differences of their sides, $\llbracket t \leq t' \rrbracket = \llbracket t < t' \rrbracket = [t] - [t']$, $\llbracket t = t' \rrbracket = |[t] - [t']|$ and $\llbracket t \neq t' \rrbracket = -|[t] - [t']|$; negation is $-x$; and conjunction and disjunction are the p -log-sum-exp and its harmonic counterpart,

$$x \wedge_p y = \frac{1}{p} \log(e^{px} + e^{py}), \quad x \vee_p y = -\frac{1}{p} \log(e^{-px} + e^{-py}), \quad (6)$$

which converge to max and min as $p \rightarrow \infty$. The derivative of a log-sum-exp is a softmax, so the gradient of a conjunction concentrates on its most violated conjunct and the gradient of a disjunction on the disjunct closest to being satisfied, with smaller p spreading the gradient over both operands [25]. The additive operations suit training, where the constraint loss is added to the data loss and network outputs are additive quantities such as logits. Vehicle allows a specification to declare its own differentiable logic by giving the interpretation of every connective and comparison, and the Vehicle tutorial [13] provides QLL in this form; Listing 4 in Appendix A reproduces the declaration. Vehicle ships its default logic and DL2, and accepts a declared logic in the same way; all experiments of this paper use QLL, with hardness $p = 5$ for ACAS Xu and $p = 4$ for the drone and dosing controllers.

3 Combined Optimisation Procedure

3.1 The Optimisation Objective

We now state the optimisation objective formally. Given:

- a neural network $f_\theta: \mathbb{R}^m \rightarrow \mathbb{R}^n$ parametrised by training parameters θ (such as neural network weights), and
- a constraint $\phi: \forall \vec{x} \in \mathbb{R}^m. \mathcal{P}(\vec{x}) \rightarrow Q(f(\vec{x}))$ which we want to incorporate into optimisation for θ ,

we define a translation function $\langle \cdot \rangle$ from a precondition $\mathcal{P}(\vec{x})$ into a hyper-rectangle as follows. Assume $\mathcal{P}(\vec{x})$ is given by an arbitrary Boolean formula containing atomic propositions only of the form $l_i \leq x_i \leq u_i$, for constants l_i, u_i , and $\vec{x} = [x_1, \dots, x_m] \in \mathbb{R}^m$ is a list of variables that denotes an input vector. Let $\vec{l}^{\min}$ and $\vec{u}^{\max}$ be constant vectors that define minimum lower and maximum upper bounds of $\vec{x}$ ⁹.

If $\mathcal{P}(\vec{x})$ is not satisfiable, we define $\langle \mathcal{P}(\vec{x}) \rangle = \emptyset$. Otherwise, we define $\langle \mathcal{P}(\vec{x}) \rangle$ by induction on the structure of $\mathcal{P}(\vec{x})$:

⁹ In neural network verification, we always assume default lower and upper bounds are inherently present (e.g. inferred from data) when some element vectors are not explicitly constrained. Examples in the rest of the paper illustrate that point.

- If $\mathcal{P}(\vec{x})$ is of the form $l_i \leq x_i \leq u_i$, we form a hyper-rectangle $H(\vec{l}^*, \vec{u}^*)$, where $\vec{l}^* = \vec{l}^{\min}$ except for its i th component being l_i (and similarly obtain $\vec{u}^*$), and define $\langle\!\langle \mathcal{P}(\vec{x}) \rangle\!\rangle = H(\vec{l}^*, \vec{u}^*)$;
- if $\mathcal{P}(\vec{x})$ is of the form $\neg F$, where F is an arbitrary formula, then $\langle\!\langle \neg F \rangle\!\rangle = \langle\!\langle F \rangle\!\rangle^\sim$;
- if $\mathcal{P}(\vec{x})$ is of the form $F_1 \vee F_2$, where F_1, F_2 are arbitrary formulas, then $\langle\!\langle F_1 \vee F_2 \rangle\!\rangle = \langle\!\langle F_1 \rangle\!\rangle \cup \langle\!\langle F_2 \rangle\!\rangle$;
- and lastly, if $\mathcal{P}(\vec{x})$ is of the form $F_1 \wedge F_2$, then $\langle\!\langle F_1 \wedge F_2 \rangle\!\rangle = \langle\!\langle F_1 \rangle\!\rangle \cap \langle\!\langle F_2 \rangle\!\rangle$;

where $\sim, \cup, \cap$ stand for set-theoretic complement, union, and intersection respectively.

The postcondition $Q(f(\vec{x}))$ is translated into a real-valued loss $\llbracket Q(f(\vec{x})) \rrbracket$ by the differentiable logic of Section 2.2.

Then, the optimisation objective for the constraint ϕ is given as follows:

$$\theta^* = \arg \min_{\theta} \mathbb{E}_{(\vec{x}, \vec{y}) \sim \mathcal{D}} \left[\alpha \mathcal{L}(\vec{x}, \vec{y}; f_{\theta}) + (1 - \alpha) \max_{\vec{x}' \in \langle\!\langle \mathcal{P}(\vec{x}) \rangle\!\rangle} \llbracket Q(f(\vec{x})) \rrbracket(\vec{x}, \vec{x}', \vec{y}; f_{\theta}) \right]. \quad (7)$$

Note the similarity to the optimisation objective previously shown in Eq. (5). However, to find adversarial examples for the constraint loss, the inner maximisation problem $\max_{\vec{x}' \in \langle\!\langle \mathcal{P}(\vec{x}) \rangle\!\rangle} \llbracket Q(f(\vec{x})) \rrbracket(\vec{x}, \vec{x}', \vec{y}; f_{\theta})$ is approximated using a modified PGD attack that works with arbitrary hyper-rectangles $\langle\!\langle \mathcal{P}(\vec{x}) \rangle\!\rangle$.

3.2 Specifications in Vehicle

The optimisation objective of Section 3.1 and the verification of the resulting network start from the same object: a specification of the property $\forall \vec{x}. \mathcal{P}(\vec{x}) \longrightarrow Q(f(\vec{x}))$. In this paper both are obtained from a single Vehicle specification. Vehicle is functional and dependently typed, and a specification is written over the *problem domain*, in the units of the application, rather than over the network’s normalised inputs. As Fig. 3 shows, one specification is compiled to several back-ends: to a loss function, in PyTorch or TensorFlow, for training the network to satisfy the property; to queries for the neural network verifier Marabou [37,70] or any VNN-LIB compliant verifier [52]; and to an interactive theorem prover (Agda, Rocq, Isabelle/HOL or Imandra), where the verified property becomes a lemma about the network in a proof about the surrounding system [17]. The compilation of quantified properties to solver queries and to losses is described in [14,15]. Marabou [37,70] is an SMT-based verifier for networks with piecewise-linear activations: given a query, linear constraints on the network’s inputs and outputs, it either proves that no input satisfies them or returns one that does, so a verdict is a proof or a counterexample. Vehicle’s `verify` command translates each property into such queries, runs Marabou on them and reports the verdict; every verification result in this paper was obtained this way, on the specification unchanged. Listing 1 shows the specification of property ϕ_2 of the ACAS Xu collision-avoidance networks [36], the running example of Section 4.1, as it appears in the Vehicle tutorial [13]: if the intruder is distant and significantly slower than the ownship, the score of the “clear of conflict” advisory is never maximal. The listing is abridged; the tutorial specification declares the remaining nine properties of [36] over the same definitions. The example exhibits the constructs used throughout this paper. A `@network` declaration gives the network’s type; the network

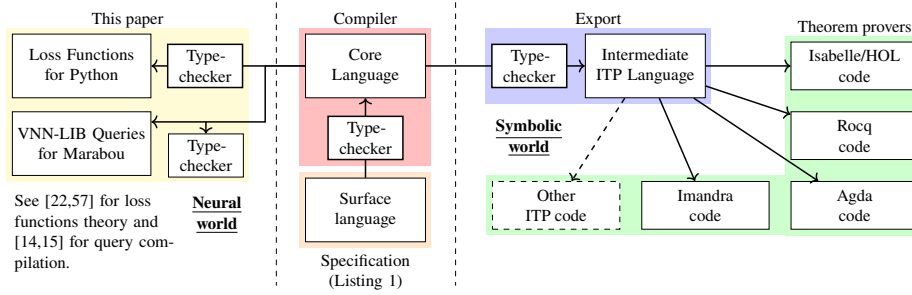


Fig. 3: The internal architecture of Vehicle, after [17,18]. A specification in the surface language is type-checked into a core language, from which a second type-checker produces either loss functions for Python and queries for Marabou (the left branch, used in this paper) or an intermediate language that is exported to an interactive theorem prover (the right branch).

itself is supplied as an ONNX file at compile time, so that the specification is independent of any particular trained network. Named indices, typed tensors and ordinary function definitions let the property be stated in the vocabulary of the application. Functions may compose the network with other functions: here `normAcasXu` applies the normalisation that the network was trained with, so that the property is stated in metres and metres per second while the network receives normalised inputs. A `@property` is a Boolean formula over quantified inputs; the quantifier ranges over the whole input type, and the antecedent of the implication restricts it, here to the box `validInput` intersected with the half-spaces of `intruderDistantAndSlower`. Two further constructs make a specification reusable. `@parameter` declarations are constants whose values are given on the command line at compile time, so that a family of properties, such as the properties of Section 4.2, whose regions depend on a distance threshold, is a single specification swept over the parameter, and constant folding turns each instance into the linear form that the verifier needs. `@dataset` declarations load tensors from files, such as normalisation statistics or the training set around whose elements a local property is to hold. The medical case study of [56] exercises all of these together, with a specification that is generic over patient parameters and is verified by Marabou and exported to Rocq from one file.

A specification of this form is exactly the input that the objective of Section 3.1 requires. When it is compiled to a loss, Vehicle extracts from the antecedent $\mathcal{P}(\vec{x})$ the hyper-rectangle $\langle \mathcal{P}(\vec{x}) \rangle$ over which the universal quantifier ranges and approximates the inner maximisation of Eq. (7) by an adversarial search over that box, while the consequent $Q(f(\vec{x}))$ is translated into a real-valued loss by a differentiable logic. The logic is the one declared in the specification (Section 2.2). The same specification, unchanged, is what Marabou verifies, so that the property trained for and the property proved are the same property, and the choice of logic, the adversarial search and the weight α affect only how well the training succeeds, never what is being verified.

```

type InputVector      = Tensor Real [5] -- what the network reads
type UnnormalisedInputVector = Tensor Real [5] -- what the property speaks about
type OutputVector     = Tensor Real [5] -- one score per advisory

distanceToIntruder = 0 -- metres          clearOfConflict = 0
angleToIntruder   = 1 -- radians          weakLeft        = 1
intruderHeading   = 2 -- radians          weakRight         = 2
speed             = 3 -- metres/second    strongLeft        = 3
intruderSpeed     = 4 -- metres/second    strongRight       = 4

@network
acasXu : InputVector -> OutputVector

minimumInputValues : UnnormalisedInputVector
minimumInputValues = [0.0, -pi, -pi, 100.0, 0.0]
maximumInputValues : UnnormalisedInputVector
maximumInputValues = [60261.0, pi, pi, 1200.0, 1200.0]
meanScalingValues : UnnormalisedInputVector
meanScalingValues = [19791.091, 0.0, 0.0, 650.0, 600.0]

validInput : UnnormalisedInputVector -> Bool
validInput x = forall i . minimumInputValues ! i <= x ! i <= maximumInputValues ! i

normalise : UnnormalisedInputVector -> InputVector
normalise x = foreach i .
  (x ! i - meanScalingValues ! i) / (maximumInputValues ! i - minimumInputValues ! i)

normAcasXu : UnnormalisedInputVector -> OutputVector
normAcasXu x = acasXu (normalise x)

maximalScore : Index 5 -> UnnormalisedInputVector -> Bool
maximalScore i x = forall j . i != j => normAcasXu x ! i > normAcasXu x ! j

intruderDistantAndSlower : UnnormalisedInputVector -> Bool
intruderDistantAndSlower x =
  x ! distanceToIntruder >= 55947.691 and
  x ! speed >= 1145 and
  x ! intruderSpeed <= 60

@property
property2 : Bool
property2 = forall x .
  validInput x and intruderDistantAndSlower x =>
  not (maximalScore clearOfConflict x)

```

Listing 1: Property ϕ_2 of ACAS Xu in Vehicle, abridged from the specification in the Vehicle tutorial [13]. The antecedent `validInput x` and `intruderDistantAndSlower x` is the precondition $\mathcal{P}(\vec{x})$ of Section 3.1, from which the hyper-rectangle $\langle \mathcal{P}(\vec{x}) \rangle$ of Eq. (8) is read off, and the consequent `not (maximalScore clearOfConflict x)` is the constraint $Q(f(\vec{x}))$.

4 Case Studies

In this section we use three case studies to demonstrate the practical applicability of the framework just described. Each reveals an aspect of specification-driven model development: training in the absence of data, specifications that contradict the data, and specifications that need refinement before they fit the hyper-rectangles of the training objective. Throughout, we keep to the principle of specification-driven training: the specification, refined where necessary, is the means of understanding and conceptualising the properties of the data and of the cyber-physical system at hand.

4.1 ACAS Xu

The Airborne Collision Avoidance System X (ACAS X) was developed at MIT Lincoln Laboratory for the US Federal Aviation Administration as the successor to TCAS: its advisories are computed offline by dynamic programming over a Markov decision process and stored in a lookup table [41,40], and ACAS Xu is its variant for unmanned aircraft. The table is too large for avionics hardware, so Julian et al. [35] compressed it into 45 neural networks, one for each combination of previous advisory and time to loss of vertical separation. Katz et al. [36] then stated ten safety properties of these networks and built the SMT-based verifier Reluplex to check them, which made ACAS Xu the first, and still the most used, benchmark for neural network verification. For this paper it is interesting for two reasons: its properties are global, stated over sensor readings with a physical meaning, and its training data were never released. Each network takes the five sensor measurements shown in Fig. 4 as inputs and produces scores for five possible advisories, “clear-of-conflict” (COC), “weak left” (WL), “weak right” (WR), “strong left” (SL), or “strong right” (SR); the advisory with the lowest score is chosen.

As a representative example, we take property ϕ_2 from [36], whose Vehicle specification is Listing 1. It requires that the score for a “clear of conflict” advisory should never be maximal for a distant, significantly slower intruder, meaning that it can never be the worst action to take.

- **Obtaining $\langle \mathcal{P} \rangle$:** The *intruder being distant and significantly slower* translates to the following lower bounds on network inputs:

$$55\,947.691 \text{ ft} \leq \rho, \quad 1145 \text{ ft/s} \leq v_{\text{own}}, \quad v_{\text{intruder}} \leq 60 \text{ ft/s}, \quad (8)$$

which directly give rise to a hyper-rectangle: assuming the inputs to the network to be a vector $\vec{x} = [\rho, \theta, \psi, v_{\text{own}}, v_{\text{intruder}}]$, the hyper-rectangle induced by Eq. (8) translates to lower bound $\vec{l}$ and upper bound $\vec{u}$ as

$$\vec{l} = \max\{\vec{l}^{\min}, [55\,947.691 \text{ ft}, -\infty, -\infty, 1145 \text{ ft/s}, 60 \text{ ft/s}]\}, \text{ and } \vec{u} = \vec{u}^{\max}, \quad (9)$$

where $\vec{l}^{\min}$ and $\vec{u}^{\max}$ denote the minimum and maximum permitted values for each of the five parameters, and $\max\{\cdot, \cdot\}$ is applied component-wise.

- **Obtaining $\langle Q(f(\vec{x})) \rangle$:** The constraint on the output, i.e. *the score for a “clear of conflict” advisory should never be maximal*, can be expressed as the disjunction

$$(y_{\text{COC}} < y_{\text{WL}}) \vee (y_{\text{COC}} < y_{\text{WR}}) \vee (y_{\text{COC}} < y_{\text{SL}}) \vee (y_{\text{COC}} < y_{\text{SR}}), \quad (10)$$

and translated into loss using the mappings $\llbracket t < t' \rrbracket$ and $\llbracket t \vee t' \rrbracket$ of a chosen differentiable logic. The actual translation is omitted here for the sake of brevity, but can be obtained by repeated application of $\llbracket t < t' \rrbracket$ and the identity $\llbracket t < t' \rrbracket = \llbracket t \leq t' \wedge t \neq t' \rrbracket$.

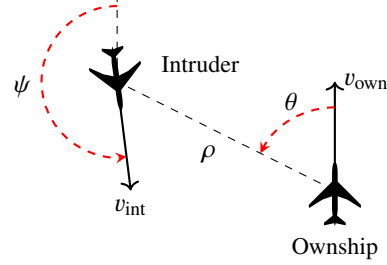


Fig. 4: The geometry of an ACAS Xu encounter: distance to the intruder ρ , angle to the intruder θ , heading of the intruder ψ , and the two speeds v_{own} and v_{int} .

Table 2: Marabou verdicts on $N_{1,8}$ before training (shipped), after one tour of the ten properties (10 epochs) and after three tours (30 epochs, optimiser re-initialised at every epoch), with the solver time; red columns are the shipped network, orange after one tour, green after three tours. ✓ verified, ✗ falsified, t/o undecided within the limit. Verified sets: $\{1, 2, 10\}$ shipped, $\{1, 2, 3, 10\}$ after one tour, $\{1, 2, 3, 4, 9, 10\}$ after three tours.

Property	shipped	one tour	three tours	Property	shipped	one tour	three tours
ϕ_1	✓ 4 s	✓ 3 s	✓ 24 s	ϕ_6	t/o 2.5 h	✗ 2 min	✗ 2 h
ϕ_2	✓ 5 min	✓ 20 s	✓ 25 s	ϕ_7	t/o 2.5 h	t/o 15 min	✗ 2 s
ϕ_3	✗ 2 s	✓ 1 s	✓ 2 s	ϕ_8	t/o 2.5 h	t/o 15 min	✗ 4 s
ϕ_4	✗ 1 s	✗ 2 s	✓ 1 s	ϕ_9	✗ 1 s	✗ 2 s	✓ 28 s
ϕ_5	✗ 1 s	✗ 1 s	✗ 24 s	ϕ_{10}	✓ 2 min	✓ 19 min	✓ 49 s

Experiment: training without data. The ACAS Xu training data have never been released, so the objective of Eq. (7) is used with $\alpha = 0$: the loss is the compiled property term alone, and the adversarial search inside it supplies every input from the hyper-rectangle of the property’s precondition. Nothing else enters training. We take the network $N_{1,8}$ and the ten properties of [36] and we also ship these as part of the Vehicle tutorial¹⁰. Of these only ϕ_1 was ever claimed to hold on $N_{1,8}$; ϕ_2 to ϕ_4 are reported not to hold on it, and ϕ_5 to ϕ_{10} were written for other networks of the family, with different previous-advisory contexts. The specification is split into ten one-property files, and training visits them in turn: epoch k takes 16 Adam steps (learning rate 10^{-3}) on the loss of property $((k - 1) \bmod 10) + 1$, with QLL at $p = 5$ and Vehicle’s default search (10 random starts, 5 PGD steps in the box). A *tour* of the ten properties takes about three minutes on a CPU. Marabou verifies every snapshot against the unchanged ten-property specification with a limit of 900 s per property; the properties still undecided at that limit were re-run with a 9000 s limit. Table 2 gives the verdicts on the shipped network, after one tour, and after three tours.

What the experiment shows.

- **Verified properties doubled, without a single data point.** The shipped network satisfies ϕ_1 , ϕ_2 and ϕ_{10} ; after one tour it also satisfies ϕ_3 , and after three tours ϕ_4 and ϕ_9 as well, six of the ten. Nothing that held was lost at any stage. The remaining four are ϕ_5 , ϕ_7 and ϕ_8 , which demand advisories $N_{1,8}$ was never trained to give, and ϕ_6 , undecided on the shipped network after 2.5 h and falsified on the trained one after 2 h.
- **The trained network is still a controller.** Training on ϕ_3 alone, in a preliminary round, verified ϕ_3 on every network in one epoch, but did so by making “clear of conflict” the minimal advisory nowhere: the networks agreed with the originals on under 1% of the input space. Ten properties pulling in different directions (ϕ_6 and ϕ_{10} demand clear-of-conflict, ϕ_7 forbids strong turns, ϕ_3 and ϕ_4 forbid clear-of-conflict) prevent this. After 30 epochs the network agrees with the shipped one on 90.5% of uniformly drawn inputs and still advises clear-of-conflict on 91% of the space; the

¹⁰ <https://github.com/vehicle-lang/tutorial/blob/exercises/chapter-4/solutions/acas2/README.md> records the full experiment, including every trace and Marabou transcript.

change is confined to the regions of ϕ_3 and ϕ_4 , where it now advises “strong right”. With no data, the other properties are the regulariser.

- **One property per epoch forgets; repetition remembers.** Within a tour, ϕ_4 and ϕ_5 hold on the snapshot trained on them and are lost by the end of the tour, each later epoch undoing part of the previous ones. Across tours every property’s loss improves monotonically from visit to visit, and the verified set grows.
- **The optimiser state matters more than clipping.** With one Adam optimiser kept across epochs, the second visit to ϕ_3 diverged: the QLL loss is unbounded below, so a satisfied property keeps inflating the weights, and the literal $\frac{1}{p} \log \sum e^{P \cdot x_i}$ overflows once a margin exceeds about $88/p$. Gradient clipping did not help, because Adam rescales the step anyway. Re-initialising the optimiser at every epoch removed the carried-over momentum and the run completed.
- **Falsifying is cheap, proving is not, and training helps the prover.** Counterexamples arrive in seconds. Proving the three whole-input-space properties ϕ_6 to ϕ_8 on the shipped network, whose outputs are nearly tied, did not finish in 2.5 h each. The trained network is decided on every property, and its wider margins prove ϕ_2 twelve times faster.

4.2 Alsomitra Drones

Nature-inspired robotics borrows its designs from organisms that accomplish a task with little or no actuation. The winged seeds of the Javan cucumber, *Alsomitra macrocarpa* (Fig. 5a), glide stably over long distances from the forest canopy on a wing a few centimetres across, and their aerodynamics have been studied as a model for passive flight [12]. The promise is a centimetre-scale microflyer, transported by the wind and steered only by shifting its centre of mass, for weather or pollution monitoring at a cost and scale that powered drones cannot reach. Kessler et al. [39] introduced a neural network controller for such a drone, trained on simulations of a two-dimensional falling-plate model, stated four global safety properties for it in Vehicle, and released the networks and the data as a verification benchmark. It is the second case study of this paper. The controller should make the drone follow the trajectory $y = -x$ in two-dimensional space (Fig. 5b); the drone is simulated with an aerodynamic model of falling plates with a displaced centre of mass [43], and the controller steers by moving that centre of mass. Its six inputs are the state of the dynamic system, the local x - and y -velocities $v_{x'}$, $v_{y'}$, the pitch angular velocity ω , the pitch angle θ and the global position x , y , normalised to $[0, 1]$ (their physical ranges are in Table 7 in Appendix A); its one output is the new position of the centre of mass, also normalised to $[0, 1]$, where values above 0.5 pitch the drone down and values below pitch it up. The benchmark ships two

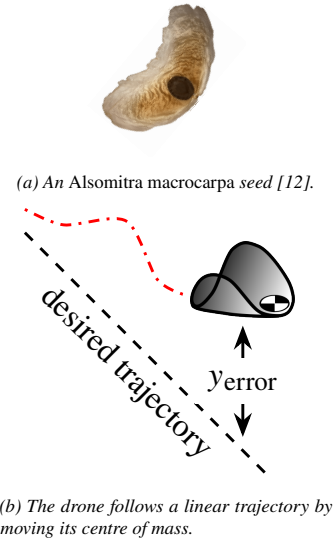


Fig. 5: The Alsomitra-inspired drone controller.

Table 3: The four global properties of the drone controller [39], stated over the flight envelope $\mathcal{E}$ and the signed distance $d(\vec{x})$ of the drone above the trajectory. The output $f(\vec{x})$ is the normalised centre-of-mass position.

	Precondition $\mathcal{P}(\vec{x})$	Postcondition $Q(f(\vec{x}))$	y^* larger
ϕ_1	$\vec{x} \in \mathcal{E}, d(\vec{x}) \geq y^*$ (far above the line)	$f(\vec{x}) \geq 0.5$ (pitch down)	easier
ϕ_2	$\vec{x} \in \mathcal{E}, d(\vec{x}) \leq -y^*$ (far below the line)	$f(\vec{x}) \leq 0.5$ (pitch up)	easier
ϕ_3	$\vec{x} \in \mathcal{E}, d(\vec{x}) \leq y^*, \theta \in [-0.796, -0.743]$ (close, intermediate pitch)	$-0.1 \leq f(\vec{x}) \leq 1.1$	harder
ϕ_4	$\vec{x} \in \mathcal{E}, d(\vec{x}) \leq y^*, \omega = \omega_{\min}, v_{y'} = v_{y', \min}$ (close, diving)	$f(\vec{x}) \leq 0.5$ (pitch up)	harder

networks of size 6-6-4-1, one with a ReLU output and one with a linear output, and the 360 state-action pairs they were trained on. This case study therefore differs from ACAS Xu in the two respects that matter for training: it is a regression task, and the data are available.

For this case study and the next, where data exist, we report the root mean squared error (RMSE) of the network on the data and, for each property and each value of its parameter, the verdict of Marabou on the unchanged specification.

Properties. The benchmark states four global properties, all parametrised by a distance y^* from the trajectory (Table 3). The first two say that the drone corrects towards the line when it is far from it, the third that the controller stays within its output range near the line at an appropriate pitch, and the fourth that a drone near the line that is already pitching and moving down fast is not pitched down further. For the first two properties a larger y^* shrinks the region and the interesting value is the smallest y^* at which the property holds; for the last two a larger y^* widens the band and the interesting value is the largest. As a representative example, we take ϕ_1 , whose Vehicle specification is Listing 2.

- **Obtaining $\langle \mathcal{P} \rangle$:** The benchmark’s own specification quantifies over the whole input box $[0, 1]^6$. Verifying it shows why that is too much: the box contains states the drone never reaches, and ϕ_1 and ϕ_2 are proved on the shipped networks only for y^* above 45.9 and 41.2, where the regions are empty and Marabou returns without a query. We therefore restrict every property to the *flight envelope* $\mathcal{E}$: the box, cut by five pairwise bounds read off the data (for instance $0.8 \leq v_{x'} + \theta \leq 1.2$) and by a band $|d(\vec{x})| \leq d_{\max}$ around the trajectory. The restriction alone does not remove the vacuity: a property $\forall \vec{x}. \mathcal{P}(\vec{x}) \rightarrow Q(f(\vec{x}))$ is true whenever $\mathcal{P}$ has no solutions, so a threshold that empties the region still yields a proof, and a universally quantified formula cannot say that its own region is inhabited. Vehicle lets a property be any Boolean combination of quantified formulae, so each property is stated as the conjunction of an existential and a universal part, $(\exists \vec{x}. \mathcal{P}(\vec{x})) \wedge (\forall \vec{x}. \mathcal{P}(\vec{x}) \rightarrow Q(f(\vec{x})))$, as in Listing 2. The first conjunct does not mention the network, so it is a linear feasibility question that Vehicle settles before any query is sent to Marabou: when the region is empty the property is reported as false without a query, and the verifier is called only for regions that contain a state. Under this form the thresholds at which the benchmark

```

@parameter ystar : Real -- distance from the trajectory
@parameter dmax  : Real -- half-width of the flight envelope around the trajectory

inEnvelope x      = validInput x and pairwiseBounds x and -dmax <= lineOffset x <= dmax
droneFarAboveLine x = lineOffset x >= ystar

@property
property1 = (exists x . inEnvelope x and droneFarAboveLine x) and
             (forall x . inEnvelope x and droneFarAboveLine x => alsomitra x ! e_x >= 0.5)

```

Listing 2: Property ϕ_1 of the drone controller in Vehicle. `lineOffset` is the signed distance $d(\vec{x})$ above the trajectory in the units of the simulation, and `pairwiseBounds` the five bounds of the flight envelope; both are defined once and shared by the four properties. The two parameters are swept from the command line.

Table 4: Marabou verdicts on the drone controllers under the envelope specification, as the number of the nine thresholds $y^* \in \{0, 0.5, 1, 1.5, 2, 3, 4, 5, 6\}$ at which each property is proved (“all” = 9), with the RMSE on the 360 data rows. The small network is the benchmark’s 6-6-4-1 controller with a linear output; the large network is 6-64-32-1, trained on the same data. Red rows are the baselines, orange the Vehicle runs on the raw, unnormalised losses, green the Vehicle runs on normalised losses.

Network	Training	RMSE	ϕ_1	ϕ_2	ϕ_3	ϕ_4
small baseline	data only	0.20	0	0	all	0
large baseline	data only	0.0095	0	0	0	0
small, Vehicle trained	raw losses, 50 epochs	0.34	0	all	all	all
large, Vehicle trained	raw losses, 60 epochs	0.31	0	8	all	all
large, Vehicle trained	normalised losses, α 0.5 to 0.2, 100 epochs	0.011	0	0	all	all
large, Vehicle trained	normalised losses, adaptive weights, RMSE budget 0.02	0.016	0	0	8	all

proved ϕ_1 and ϕ_2 become failures, which is the honest verdict. For training the guard is dropped, since a region that is empty has nothing to train on and a region that is not contributes no gradient through it; the compiled loss is that of the universal part alone. The envelope is a polytope, not a hyper-rectangle, so it is not of the form that Section 3.1 assumes and that Vehicle’s loss compiler extracts. For training, each region is mapped onto the unit cube coordinate by coordinate, $\vec{u} \in [0, 1]^6 \mapsto \vec{x}(\vec{u})$, with the bounds of each coordinate depending on the ones already fixed, so that $\mathcal{P}$ becomes the box $[0, 1]^6$ in $\vec{u}$; the map is exact, and the verifier still uses the original polytope.

- **Obtaining $\llbracket Q(f(\vec{x})) \rrbracket$** : The postcondition is the atom $f(\vec{x}) \geq 0.5$, whose QLL loss is the margin $0.5 - f(\vec{x})$.

Experiment: training with, and against, data. Here the objective of Eq. (7) is used in full: the task loss is the mean squared error on the data, and the four property losses, compiled by Vehicle from the envelope specification with QLL at $p = 4$, are summed at

every step rather than visited in turn, which is the lesson of Section 4.1. The task and property terms are of incomparable size (the mean squared error of the baselines is about 10^{-4} , the property losses of order 1), so each term is divided by its value on the starting network, and α is annealed from 0.5 to 0.2 over 100 epochs of 30 Adam steps at learning rate 10^{-3} . Vehicle’s default adversarial search missed violations that lie at the corners of the envelope, so the search keeps a buffer of the worst points found for each property, starts from the box corners, and takes the most violating trajectory rather than the least. Every tenth snapshot is verified by Marabou at nine thresholds; both networks were also trained with the raw, unnormalised losses for comparison. Beyond the benchmark’s small network we train a large one, 6-64-32-1, on the same data, because with ten hidden units the output of the small network moves as one block under the property losses. Table 4 gives the verdicts.

What the experiment shows.

- **Specification refinement may prove to be necessary.** Initially, the thresholds at which the benchmark proved ϕ_1 and ϕ_2 were the thresholds at which their regions become empty. Restricting the properties to the flight envelope and guarding them against emptiness turned two proofs into failures on every network, before any training.
- **Specifications may replace, not just complement, the data.** Under the normalised objective the large network proves ϕ_3 and ϕ_4 at all nine thresholds from epoch 70 on, at an RMSE of 0.011 against 0.0095 before training. The region of ϕ_4 contains no data at all; its behaviour there is supplied by the property alone.
- **Verified is not the same as useful: the model must remain a useful controller after specification-driven training.** The one network that proves three properties, the small one trained on raw losses, is a constant: it outputs 0.47 on every data row, with the fit destroyed. In Section 4.1 the other nine properties were the regulariser because there were no data; here the data are the regulariser, and normalising the two terms is what lets them act. One needs to ensure the right balance in data-driven and specification-driven training.
Since the RMSE measures the fit to the data (smaller is better), one may prefer the large Vehicle-trained network to the small one on that ground alone, at the price of ϕ_2 , which the large one does not verify.
- **The two properties that stay unverified are the two that contradict the data.** Below $y^* \approx 3$ the recorded controller pitches up where ϕ_1 demands pitch down, and just below the line it outputs up to 0.2 where ϕ_2 allows at most 0.5 with no margin. Their losses fall throughout the run and never cross zero. Properties may deviate from the data, that is the point of the exercise, but then the property has to be allowed to overrule the data on those rows. Giving each property its own weight, raised by dual ascent while it is violated, with α adapted to hold the RMSE at 0.02, verifies ϕ_3 and ϕ_4 forty epochs sooner but not ϕ_1 or ϕ_2 : their weights sit at the ceiling of ten times the others from epoch 20 on, and their losses end at 0.30 and 0.33, the lowest of any run, still short of zero. Within a fit budget of 0.02 the data win.
There is a choice at this point, whether to let the specification win over the data or not, and it depends on the scenario in question.

- **The search matters.** On the starting network the default search reported ϕ_1 and ϕ_3 as satisfied while the corner search found violations of +0.5 and +0.9; a ‘forall’ that keeps the least violating trajectory trains against an optimistic loss. This is a case where the specification informs how the optimiser itself is tuned.

4.3 Medical Controllers

Vancomycin is one of the most widely prescribed antibiotics in hospitals, a first-line treatment for severe Gram-positive infections with a narrow therapeutic window: too high a concentration risks irreversible kidney damage, too low risks treatment failure, and the right dose varies with the patient’s weight, age and state of infection. Dosing is adjusted today by therapeutic drug monitoring, a model-dependent and human-mediated process that misses its targets routinely. A neural network controller that adapts the dose to the patient’s state is an attractive alternative, however an unverified one is a risk at scale. Sirman et al. [56] introduced Vancomycert, to our knowledge the first neural dosing controller verified end to end: a property of the network, checked by Marabou through Vehicle, is imported into Rocq as an axiom and there shown to imply that the drug concentration never exceeds the safe bound over an infinite horizon, parameterised for each patient. What makes it interesting here is that the specification is generic over the patient’s pharmacokinetic parameters and that the pharmacokinetic model appears inside the property.

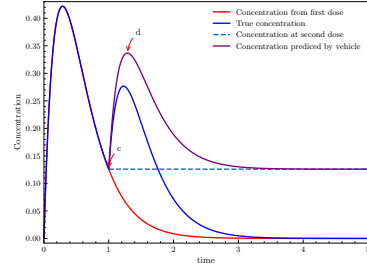


Fig. 6: Two doses in the one-compartment model: first dose alone (red), true total (blue), and the sum of the single-dose contributions that the property uses (purple). c is the concentration at the second decision, d the predicted peak.

The system and its property. The controller reads five patient measurements, the current drug concentration c , body temperature, white blood cell count, age and weight, and returns the next dose D , administered at regular intervals. The body is modelled as a one-compartment pharmacokinetic system [60]: after a dose D the concentration it contributes is

$$C(D, t) = \frac{D \cdot k_a}{Vd \cdot (k_a - k_e)} \cdot (e^{-k_e t} - e^{-k_a t}), \quad (11)$$

with absorption constant k_a , elimination constant k_e and volume of distribution Vd ; the red curve of Fig. 6, taken from [56], is Eq. (11) for one dose. The total concentration after several doses is taken to be the sum of their contributions (the purple curve), which over-approximates the true concentration (blue), since the body eliminates faster at higher concentrations; the over-approximation is what makes the safety argument sound. The safety property is that this total never exceeds the supratherapeutic bound C_{safe} , for any number of doses. Sirman et al. prove in Rocq that it follows from a property of the controller at a single decision: if the current concentration c is below C_{safe} , then c plus the *peak* of Eq. (11) for the chosen dose, reached at $t^* = \ln(k_a/k_e)/(k_a - k_e)$, stays below C_{safe} . This property is linear in the dose, and therefore in the network, once

Table 5: The three properties of the dosing controller [56], over the box $\mathcal{B}$ of clinically viable patients (concentration 0 to C_{safe} , temperature 36.5 to 40, white cells 7.5 to 20, age 18 to 89, weight 50 to 100). $D = f(\vec{x})$ is the dose; $\kappa = k_a / (Vd(k_a - k_e)) \cdot (K_{\text{e_over}} - K_{\text{a_under}})$ is the bracketed peak per unit dose.

	Precondition $\mathcal{P}(\vec{x})$	Postcondition $Q(f(\vec{x}))$
safeFar	$\vec{x} \in \mathcal{B}, c \leq 0.99 C_{\text{safe}}$	$c + \kappa D < C_{\text{safe}}$ (the next peak is safe)
safeNear	$\vec{x} \in \mathcal{B}, 0.99 C_{\text{safe}} \leq c \leq C_{\text{safe}}$	$D < \epsilon$ (do not dose at the limit)
nonNeg	$\vec{x} \in \mathcal{B}$	$D > 0$ (needed by the Rocq proof)

```

@parameter Ka : Real      -- absorption constant
@parameter Ke : Real      -- elimination constant
@parameter Vd : Real      -- volume of distribution
@parameter Ke_over : Real
@parameter Ka_under : Real
@parameter C_safe : Real

safeFarInput : InputVector -> Bool
safeFarInput x =
  0 <= x ! conc <= C_safe * 0.99 and 36.5 <= x ! temp <= 40 and 7.5 <= x ! wbc <= 20 and
  18 <= x ! age <= 89 and 50 <= x ! weight <= 100

safeFarOutput : InputVector -> Bool
safeFarOutput x =
  (x ! conc) + ((normpk x ! 0) * Ka / (Vd * (Ka - Ke))) * (Ke_over - Ka_under) < C_safe

@property
safeFar = forall x . safeFarInput x => safeFarOutput x

```

Listing 3: The property `safeFar` of the dosing controller in *Vehicle*, abridged from [56]. `normpk` applies the network to the normalised patient state, and the parameters instantiate the pharmacokinetic model of one patient population.

the two exponentials at t^* are replaced by rational brackets $[K_{\text{e_under}}, K_{\text{e_over}}]$ and $[K_{\text{a_under}}, K_{\text{a_over}}]$ that are passed to *Vehicle* as parameters and checked in Rocq; for $k_a > k_e$ the peak is over-approximated by the upper end of the first bracket and the lower end of the second, which is the form in Listing 3. Table 5 lists the three properties, and Listing 3 shows `safeFar`, the representative one, in *Vehicle*; the whole specification is generic over the patient model, whose constants k_a , k_e , Vd , C_{safe} and the brackets are `@parameters`, and normalises its inputs with two `@datasets` of statistics.

- **Obtaining $\llbracket \mathcal{P} \rrbracket$:** The precondition is a box in the five measurements, so it is a hyper-rectangle as it stands, with the concentration coordinate cut at $0.99 C_{\text{safe}}$; no reparametrisation is needed. The region of `safeNear` lies entirely outside the training data, whose concentrations never exceed 6.8, and the box of `safeFar` contains the data only in its lowest corner.
- **Obtaining $\llbracket Q(f(\vec{x})) \rrbracket$:** The postcondition is one atom, linear in the dose, whose QLL loss is the margin $c + \kappa f(\vec{x}) - C_{\text{safe}}$, in units of concentration.

Experiment: training on data the properties never see. The baseline is the controller of [56] trained on data alone and exported without its positivity shift, a 5-128-64-1 network with a ReLU output trained on 2350 synthetic clinician-style dosing decisions,

Table 6: Marabou verdicts on the dosing controller of [56] (5-128-64-1, ReLU output, no positivity shift) before and during Vehicle training, with the RMSE on the 2350 dosing rows in mg. ✓ verified, ✗ falsified. Red is the baseline, orange the run with one of the two safety properties verified, green with both. The RMSE is measured on the data alone.

Network	Training	RMSE	safeFar	safeNear	nonNeg
baseline	data only	10.8	✗	✗	✗
Vehicle trained	epoch 30	9.0	✓	✗	✗
Vehicle trained	epoch 50	7.0	✓	✓	✗
Vehicle trained	epoch 100	4.9	✓	✓	✗

which fails all three properties as delivered: it overdoses a light, young patient already at concentration 18, still prescribes at the safe limit, and returns a dose of exactly zero for one patient. The parameters are those of [56]: $k_a = 4.5$, $k_e = 3.5$, $Vd = 10$, $C_{\text{safe}} = 60$, $\epsilon = 0.001$ mg, and brackets $[0.4149, 0.415]$ and $[0.3227, 0.3228]$, so that a dose D raises the predicted peak by $0.0415 D$. The regime is that of Section 4.2: the mean squared error on the dosing data and the three property losses, compiled by Vehicle with QLL at $p = 4$, summed at every step with both terms normalised, α annealed from 0.5 to 0.2 over 100 epochs of 30 steps, the corner-and-buffer search, and Marabou on every tenth snapshot against the unchanged specification; positivity is trained with a margin of 1 mg, since a strict inequality has no margin of its own. An epoch takes five seconds; Table 6 gives the verdicts.

What the experiment shows.

- **Specifications supply what the data lack, as in the two earlier studies.** safeFar holds from epoch 30 and safeNear from epoch 50, to the end of the run, while the RMSE halves from 10.8 to 4.9 mg. Every violation Marabou found lies at a concentration above 18, outside the data, so no data row contradicts the properties and the data term keeps optimising while they are learnt. In Section 4.1 there were no data at all and the other properties regularised; in Section 4.2 the data were present and, for two properties, in conflict. This study is the case in between: data present, but silent where the properties speak, the situation of the drone’s ϕ_4 without its ϕ_1 .
- **Verified and, this time, useful.** The losses settle at exactly the values of a zero dose: with a ReLU output the cheapest way to dose less is to dose nothing, and the trained controller prescribes 0 at high concentrations. In Section 4.2 the same collapse to a constant destroyed the fit; here the fit improves, because a zero dose near the safe limit is what the clinician’s rule would also give if the data reached there. Whether a constant is a defect depends on where it is constant.
- **What the loss cannot see, training cannot fix.** The loss of nonNeg stays at +1.0, the full margin, with zero gradient at every epoch: the offending patient is where the output ReLU is off, the loss reports the violation but nothing can move it, and Marabou refutes the property on every snapshot. The property itself is easily met, and the authors of [56] met it with a constant shift of the output at export; what fails is the loss as a signal. This is the counterpart of the optimiser lesson of Section 4.1

and the search lesson of Section 4.2: the objective, the search and the network’s own activations all decide what specification-driven training can reach.

- **The specification needs refinement, as the drone’s did.** `safeNear` allows a dose below $\epsilon = 0.001$ mg at the limit and `nonNeg` demands a positive dose everywhere, a window of one microgram that a ReLU output cannot hit. The Rocq proof needs only non-negativity, so the specification, not the training, is what to change, which is the first lesson of Section 4.2 again.

5 Conclusions

We have argued, through three case studies, that a specification written in Vehicle is the one artefact from which both training and verification follow. ACAS Xu showed specification-driven training doubling the number of verified properties of a shipped network with no data at all, the other properties acting as the regulariser. The Alsomitra drone showed a specification that had to be refined before it said what its authors meant, properties supplying behaviour where the data are silent, and a genuine conflict between property and data that a per-property weight can expose but not decide. The dosing controller showed properties learnt entirely outside the data at no cost to the fit, and a positivity property that no loss can see through a ReLU floor. Across the three, the same objective, the same logic and the same search were used, and Marabou always checked the specification unchanged. The models became more verifiable, and therefore more safe (in terms of their safety specifications) in every case; the cost in prediction performance reported in [62] appeared only when the data and property terms were left unnormalised.

5.1 Related Work

Other specification languages. Similar to Vehicle, the CAISAR [28,5,33] platform provides a higher-level specification interface based on WhyML and translates verification problems to multiple back-end solvers. On the contrary, CAISAR does not compile specifications into loss functions for use in specification-driven training.

Specification-driven training. Some of the ideas presented in this paper were already present in DL2 [22]. However, while arbitrary box constraints (i.e. hyper-rectangles) are mentioned, no rationale is provided explaining why these might prove to be useful, and the experimental evaluation is limited to ϵ -balls.

Apart from DL2, various other approaches have explored integrating logical constraints into the training process: Semantic-based Regularisation (SBR) [19] encodes first order logic rules as additional loss terms. Logic Tensor Networks [53] adopt a framework called Real Logic, where symbols and predicates are grounded in real-valued vector spaces such that logical constraints can be integrated directly into the neural network’s architecture. Furthermore, probabilistic differentiable logics such as DeepProbLog [47] and Semantic Loss [71] combine logical reasoning with probabilistic inference.

Frameworks for specification-driven training. Several existing frameworks aim to provide a unified view of specification-driven training.

Flinkow et al. [24,23] present a reusable, extensible Python framework for specification-driven machine learning that unifies the following components in a modular implementation: (1) input-region specifications via hyper-rectangles, (2) adversarial specification-driven training with Projected Gradient Descent (PGD), and (3) multiple differentiable logics for encoding logical specifications as loss terms. PYLON [2] is a Python framework for specification-driven training, however, its scope is limited to output-level constraints, without employing PGD or other methods to find adversarial examples that violate the constraints. Similarly, ULLER [65] is a unifying language for neuro-symbolic AI that allows for fuzzy, classical, and probabilistic semantics. An implementation does not yet exist and is left for future work.

Architectures that guarantee constraint satisfaction. Giunchiglia et al. [29] provide a survey of specification-driven machine learning, including specialised neural network architectures to obtain hard guarantees of constraint satisfaction. Recent efforts have focused on integrating logical constraints in a neural network layer such that the network is guaranteed to satisfy the constraints by design. For example, CCN+ [30] integrates constraints into the output layer of the network, while the Semantic Probabilistic Layer (SPL) [3] leverages probabilistic circuits to enable exact probabilistic inference with logical reasoning. More recent work has focused on non-convex constraints: the Disjunctive Refinement Layer (DRL) [58] supports non-convex and disconnected spaces, and the Probabilistic Algebraic Layer (PAL) [42] guarantees satisfaction of non-convex algebraic constraints.

Limitations and future work. Neural network verification was shown to be an NP-complete problem [36], which puts a theoretical limit on the scalability of verification tools. We thus conjecture that the methodology we champion is best used on neural networks of medium size, up to several dozens of millions of parameters [20]. Specification-driven training, however, may yield further optimisations and improvements in the future, and may scale to larger neural networks.

While they are computationally effective, hyper-rectangles are not the most precise geometric representation of an input region. The flight envelope of the Alsomitra drone is a polytope, with bounds on some coordinates that depend on others; in Section 4.2 it was handled by an exact map of the unit cube onto each region, written by hand for that specification. A general treatment of polytopes in the loss compiler, or an adversarial search that works over them directly, would remove that step. Casadio et al. [10] note that a convex hull would capture input regions the most accurately, however, constructing this would be computationally expensive.

On the other hand, loss-based methods provide (almost) no formal guarantees, as confirmed in [26]. Worth exploring in future work is to investigate whether ideas from certified training [68,69,50,51] can be integrated into this framework.

An interesting subject for further study is to define in more precise terms the fragment of FOL that can be covered by the admissible verification properties that correspond to optimisation objectives. The property $\forall \vec{x} \in \mathbb{R}^m. \mathcal{P}(\vec{x}) \longrightarrow Q(f(\vec{x}))$ clearly has a clausal form, and there is a long history of studying first-order Horn clauses and first-order

hereditary Harrop clauses as optimal fragments of FOL for automated reasoning and verification [49,6].

Acknowledgments. Flinkow acknowledges the support of Taighde Éireann – Research Ireland grant number 20/FFP-P/8853 and the ADAPT Research Ireland Centre for AI Driven Digital Content Technology at Maynooth University under grant 13/RC/2106_P2. Komendantskaya acknowledges the partial support of the EPSRC grant AISEC: AI Secure and Explainable by Construction (EP/T026960/1), and support of ARIA: Mathematics for Safe AI grant. Kessler is funded by the EPSRC CDT “Edinburgh Centre for Robotics”. Casadio acknowledges a James Watt PhD Scholarship from Heriot-Watt University. Sirman was funded by UKRI Industrial Partnership PhD scholarship and SLB Cambridge Research.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

A Appendix

A.1 The Additive QLL as a Vehicle Logic

Listing 4 is the declaration of the additive QLL of Eq. (6) as a differentiable logic in Vehicle, used in all experiments of this paper (with hardness $p = 5$ for ACAS Xu and $p = 4$ for the drone and dosing controllers).

A.2 Alsomitra Drone Controller Inputs

Table 7 describes the 6 inputs to the Alsomitra drone controller with their intended meaning, unit, and lower and upper bounds.

Table 7: The Alsomitra drone controller inputs and their ranges.

Parameter	Description	Unit	Range	
			Lower	Upper
$v_{x'}$	local x -velocity	m/s	0.97	3.49
$v_{y'}$	local y -velocity	m/s	-0.61	-0.04
ω	pitch angular velocity	rad	-0.36	0.05
θ	pitch angle	rad	-0.97	-0.07
x	global x -position	m	0.48	41.71
y	global y -position	m	-41.68	4.20

```

p : Real
p = 4.0

qllAdditive : DifferentiableTensorLogic
qllAdditive =
{ trueElement      = -infinity
, falseElement     = infinity
, pointwiseNegation = \x -> -x
, pointwiseConjunction = \{dims} x y ->
  (const (1/p) dims) * log(exp(const p dims * x) + exp(const p dims * y))
, pointwiseDisjunction = \{dims} x y ->
  -(const (1/p) dims) * log(exp(const (-p) dims * x) + exp(const (-p) dims * y))
, pointwiseLessThan = \x y -> x - y
, pointwiseLessEqualThan = \x y -> x - y
, pointwiseGreaterThan = \x y -> y - x
, pointwiseGreaterEqualThan = \x y -> y - x
, pointwiseEqual = \x y -> max (x - y) (y - x)
, pointwiseNotEqual = \x y -> - max (x - y) (y - x)
, reduceConjunction = \{dims} xs -> (1/p) * log(reduceAdd (exp (const p dims * xs)))
, reduceDisjunction = \{dims} xs -> -(1/p) * log(reduceAdd (exp (const (-p) dims * xs)))
}

```

Listing 4: The additive QLL of Eq. (6) declared as a differentiable logic in `Vehicle`, as in the `Vehicle` tutorial [13]. The `reduce` fields interpret the quantifiers over tensor indices, and `const` broadcasts a scalar to the shape `dims` of its operands.

References

1. Affeldt, R., Bruni, A., Komendantskaya, E., Ślusarz, N., Stark, K.: Taming Differentiable Logics with Coq Formalisation (Mar 2024). <https://doi.org/10.48550/arXiv.2403.13700>
2. Ahmed, K., Li, T., Ton, T., Guo, Q., Chang, K.W., Kordjamshidi, P., Srikumar, V., den Broeck, G.V., Singh, S.: PYLON: A PyTorch Framework for Learning with Constraints. *Proceedings of the AAAI Conference on Artificial Intelligence* **36**(11), 13152–13154 (Jun 2022). <https://doi.org/10.1609/aaai.v36i11.21711>
3. Ahmed, K., Teso, S., Chang, K.W., den Broeck, G.V., Vergari, A.: Semantic Probabilistic Layers for Neuro-Symbolic Learning. In: *Advances in Neural Information Processing Systems* (Oct 2022)
4. Albarghouthi, A.: Introduction to Neural Network Verification (Oct 2021). <https://doi.org/10.48550/arXiv.2109.10317>
5. Alberti, M., Bobot, F., Girard-Satabin, J., Grastien, A., Varasse, A., Chihani, Z.: The CAISAR Platform: Extending the Reach of Machine Learning Specification and Verification. In: Damiani, F., Farrell, M. (eds.) *Integrated Formal Methods*. pp. 290–309. Springer Nature Switzerland, Cham (2026). https://doi.org/10.1007/978-3-032-10794-7_15
6. Bjørner, N., Gurfinkel, A., McMillan, K., Rybalchenko, A.: Horn Clause Solvers for Program Verification. In: Beklemishev, L.D., Blass, A., Dershowitz, N., Finkbeiner, B., Schulte, W. (eds.) *Fields of Logic and Computation II*, vol. 9300, pp. 24–51. Springer International Publishing, Cham (2015). https://doi.org/10.1007/978-3-319-23534-9_2
7. Bunel, R., Turkaslan, I., Torr, P., Pawan Kumar, M., Lu, J., Kohli, P.: Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research* **21** (2020)
8. Capucci, M., Atkey, R., Grellois, C., Komendantskaya, E., Daggitt, M.L.: Adequate Losses via Quantitative Linear Logic (May 2026). <https://doi.org/10.48550/arXiv.2605.13348>
9. Casadio, M., Arnaboldi, L., Daggitt, M., Isac, O., Dinkar, T., Kienitz, D., Rieser, V., Komendantskaya, E.: ANTONIO: Towards a Systematic Method of Generating NLP Bench-

- marks for Verification. In: Proceedings of the 6th Workshop on Formal Methods for ML-Enabled Autonomous Systems. Kalpa Publications in Computing, vol. 16, pp. 59–70 (2023). <https://doi.org/10.29007/7wxb>
10. Casadio, M., Dinkar, T., Komendantskaya, E., Arnaboldi, L., Daggitt, M.L., Isac, O., Katz, G., Rieser, V., Lemon, O.: NLP verification: Towards a general methodology for certifying robustness. *European Journal of Applied Mathematics* pp. 1–58 (Apr 2025). <https://doi.org/10.1017/S0956792525000099>
 11. 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: Shoham, S., Vizel, Y. (eds.) *Computer Aided Verification*. pp. 219–231. *Lecture Notes in Computer Science*, Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-031-13185-1_11
 12. Certini, D.: Flight of Alsomitra Macrocarpa. Ph.D. thesis, University of Edinburgh (Feb 2023). <https://doi.org/10.7488/era/3088>
 13. Daggitt, M., Kokke, W., Komendantskaya, E., Atkey, R., Arnaboldi, L., Slusarz, N., Casadio, M., Coke, B., Lee, J.: The Vehicle Tutorial: Neural Network Verification with Vehicle. In: Kalpa Publications in Computing. vol. 16, pp. 1–5. *EasyChair* (Oct 2023). <https://doi.org/10.29007/5s2x>
 14. Daggitt, M.L., Atkey, R., Kokke, W., Komendantskaya, E., Arnaboldi, L.: Compiling Higher-Order Specifications to SMT Solvers: How to Deal with Rejection Constructively. In: Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs. pp. 102–120. CPP 2023, Association for Computing Machinery, New York, NY, USA (Jan 2023). <https://doi.org/10.1145/3573105.3575674>
 15. Daggitt, M.L., Kokke, W., Atkey, R.: Efficient compilation of expressive problem space specifications to neural network solvers (Jan 2024). <https://doi.org/10.48550/arXiv.2402.01353>
 16. Daggitt, M.L., Kokke, W., Atkey, R., Arnaboldi, L., Komendantskaya, E.: Vehicle: Interfacing Neural Network Verifiers with Interactive Theorem Provers (Feb 2022)
 17. Daggitt, M.L., Kokke, W., Atkey, R., Komendantskaya, E., Slusarz, N., Arnaboldi, L.: Vehicle: Bridging the Embedding Gap in the Verification of Neuro-Symbolic Programs. In: Fernández, M. (ed.) *10th International Conference on Formal Structures for Computation and Deduction (FSCD 2025)*. *Leibniz International Proceedings in Informatics (LIPIcs)*, vol. 337, pp. 2:1–2:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2025). <https://doi.org/10.4230/LIPIcs.FSCD.2025.2>
 18. Daggitt, M.L., Komendantskaya, E., Sirman, A., Bruni, A., Teuber, S., Smart, J., Passmore, G.: Compositional Neural-Cyber-Physical System Verification in the Interactive Theorem Prover of Your Choice. *Proceedings of the ACM on Programming Languages* **10**(ICFP), 204–235 (2026)
 19. Diligenti, M., Gori, M., Saccà, C.: Semantic-based regularization for learning and inference. *Artificial Intelligence* **244**, 143–165 (Mar 2017). <https://doi.org/10.1016/j.artint.2015.08.011>
 20. Duong, H., Nguyen, T., Dwyer, M.B.: NeuralSAT: A High-Performance Verification Tool for Deep Neural Networks. In: Piskac, R., Rakamarić, Z. (eds.) *Computer Aided Verification*. pp. 409–423. Springer Nature Switzerland, Cham (2025). https://doi.org/10.1007/978-3-031-98679-6_19
 21. Farrell, M., Mavridou, A., Schumann, J.: Exploring Requirements for Software that Learns: A Research Preview. In: Ferrari, A., Penzenstadler, B. (eds.) *Requirements Engineering: Foundation for Software Quality*. pp. 179–188. *Lecture Notes in Computer Science*, Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-29786-1_12
 22. Fischer, M., Balunovic, M., Drachler-Cohen, D., Gehr, T., Zhang, C., Vechev, M.: DL2: Training and Querying Neural Networks with Logic. In: Proceedings of the 36th International Conference on Machine Learning. pp. 1931–1941. PMLR (May 2019)

23. Flinkow, T., Casadio, M., Kessler, C., Monahan, R., Komendantskaya, E.: A General Framework for Property-Driven Machine Learning (Jun 2025). <https://doi.org/10.48550/arXiv.2505.00466>
24. Flinkow, T., Grinbergs, G.G.: A general framework for property-driven machine learning. <https://github.com/property-driven-ml/property-driven-ml> (2025)
25. Flinkow, T., Komendantskaya, E., Capucci, M., Monahan, R.: Quantitative Linear Logic for Neuro-Symbolic Learning and Verification (May 2026). <https://doi.org/10.48550/arXiv.2605.13845>
26. Flinkow, T., Pearlmutter, B.A., Monahan, R.: Comparing differentiable logics for learning with logical constraints. *Science of Computer Programming* **244**, 103280 (Sep 2025). <https://doi.org/10.1016/j.scico.2025.103280>
27. Flood, R., Casadio, M., Aspinall, D., Komendantskaya, E.: Formally Verifying Robustness and Generalisation of Network Intrusion Detection Models. In: *Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing. SAC '25*, Association for Computing Machinery, New York, NY, USA (Apr 2025). <https://doi.org/10.1145/3672608.3707927>
28. Girard-Satabin, J., Alberti, M., Bobot, F., Chihani, Z., Lemesle, A.: CAISAR: A platform for Characterizing Artificial Intelligence Safety and Robustness. In: *CEUR-Workshop Proceedings. CEUR-Workshop Proceedings*, Vienne, Austria (Jul 2022)
29. Giunchiglia, E., Stoian, M.C., Lukasiewicz, T.: Deep Learning with Logical Constraints. In: *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*. pp. 5478–5485. International Joint Conferences on Artificial Intelligence Organization, Vienna, Austria (Jul 2022). <https://doi.org/10.24963/ijcai.2022/767>
30. Giunchiglia, E., Tatomir, A., Stoian, M.C., Lukasiewicz, T.: CCN+: A neuro-symbolic framework for deep learning with requirements. *Int. J. Approx. Reasoning* **171**(C) (Aug 2024). <https://doi.org/10.1016/j.ijar.2024.109124>
31. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and Harnessing Adversarial Examples (Mar 2015). <https://doi.org/10.48550/arXiv.1412.6572>
32. Goyal, S., Dvijotham, K., Stanforth, R., Bunel, R., Qin, C., Uesato, J., Arandjelovic, R., Mann, T.A., Kohli, P.: Scalable Verified Training for Provably Robust Image Classification. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. pp. 4841–4850 (Oct 2019). <https://doi.org/10.1109/ICCV.2019.00494>
33. Grastien, A., Ardouin, G., Girard-Satabin, J.: Principled Rewriting of ONNX Operators for Reluctant Solvers. In: Avni, G., Schilling, C. (eds.) *AI Verification*. pp. 59–72. Springer Nature Switzerland, Cham (2027). https://doi.org/10.1007/978-3-032-32357-6_3
34. Huang, X., Kroening, D., Ruan, W., Sharp, J., Sun, Y., Thamo, E., Wu, M., Yi, X.: A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Computer Science Review* **37**, 100270 (Aug 2020). <https://doi.org/10.1016/j.cosrev.2020.100270>
35. Julian, K.D., Lopez, J., Brush, J.S., Owen, M.P., Kochenderfer, M.J.: Policy compression for aircraft collision avoidance systems. In: *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*. pp. 1–10 (Sep 2016). <https://doi.org/10.1109/DASC.2016.7778091>
36. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In: Majumdar, R., Kunčák, V. (eds.) *Computer Aided Verification*. pp. 97–117. Lecture Notes in Computer Science, Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_5
37. Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., Dill, D.L., Kochenderfer, M.J., Barrett, C.: The Marabou Framework for Verification and Analysis of Deep Neural Networks. In: Dillig, I., Tasiran, S. (eds.) *Computer Aided Verification*. pp. 443–452. Lecture Notes in Computer Science, Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_26

38. Kaulen, K., Ladner, T., Bak, S., Brix, C., Duong, H., Flinkow, T., Johnson, T.T., Koller, L., Manino, E., Nguyen, T.H., Wu, H.: The 6th International Verification of Neural Networks Competition (VNN-COMP 2025): Summary and Results (Dec 2025). <https://doi.org/10.48550/arXiv.2512.19007>
39. Kessler, C., Komendantskaya, E., Casadio, M., Viola, I.M., Flinkow, T., Othman, A.A., Malhotra, A., McPherson, R.: Neural Network Verification for Gliding Drone Control: A Case Study. In: Giacobbe, M., Lukina, A. (eds.) *AI Verification*. pp. 180–199. Springer Nature Switzerland, Cham (2026). https://doi.org/10.1007/978-3-031-99991-8_9
40. Kochenderfer, M.J., Amato, C., Chowdhary, G., How, J.P., Reynolds, H.J.D., Thornton, J.R., Torres-Carrasquillo, P.A., Ure, N.K., Vian, J.: *Optimized Airborne Collision Avoidance*. In: *Decision Making Under Uncertainty: Theory and Application*, pp. 249–276. MIT Press (2015)
41. Kochenderfer, M.J., Chryssanthacopoulos, J.P.: *Robust Airborne Collision Avoidance through Dynamic Programming*. Tech. Rep. ATC-371, Massachusetts Institute of Technology, Lincoln Laboratory (Jan 2011)
42. Kurscheidt, L., Moretini, P., Sebastiani, R., Passerini, A., Vergari, A.: A Probabilistic Neuro-symbolic Layer for Algebraic Constraint Satisfaction (Jun 2025). <https://doi.org/10.48550/arXiv.2503.19466>
43. Li, H., Goodwill, T., Jane Wang, Z., Ristroph, L.: Centre of mass location, flight modes, stability and dynamic modelling of gliders. *Journal of Fluid Mechanics* **937**, A6 (Apr 2022). <https://doi.org/10.1017/jfm.2022.89>
44. Liu, C., Arnon, T., Lazarus, C., Strong, C., Barrett, C., Kochenderfer, M.J.: Algorithms for Verifying Deep Neural Networks. *Foundations and Trends in Optimization* **4**(3–4), 244–404 (Feb 2021). <https://doi.org/10.1561/24000000035>
45. Lopez, D.M., Choi, S.W., Tran, H.D., Johnson, T.T.: NNV 2.0: The Neural Network Verification Tool. In: Enea, C., Lal, A. (eds.) *Computer Aided Verification*. pp. 397–412. Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-37703-7_19
46. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards Deep Learning Models Resistant to Adversarial Attacks. In: *International Conference on Learning Representations* (Feb 2018)
47. Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., De Raedt, L.: DeepProbLog: Neural Probabilistic Logic Programming. In: *Advances in Neural Information Processing Systems*. vol. 31. Curran Associates, Inc. (2018)
48. Manzananas Lopez, D., Althoff, M., Benet, L., Blab, C., Forets, M., Jia, Y., Johnson, T.T., Kranzl, M., Ladner, T., Linauer, L., Neubauer, P., Neubauer, S., Schilling, C., Zhang, H., Zhong, X.: ARCH-COMP24 Category Report: Artificial Intelligence and Neural Network Control Systems (AINNCS) for Continuous and Hybrid Systems Plants. In: *Proceedings of the 11th Int. Workshop on Applied Verification for Continuous and Hybrid Systems*. pp. 64–5 (2024). <https://doi.org/10.29007/mxld>
49. Miller, D., Nadathur, G.: *Programming with Higher-Order Logic*. Cambridge University Press, 1 edn. (Jun 2012). <https://doi.org/10.1017/CBO9781139021326>
50. Mirman, M., Gehr, T., Vechev, M.: Differentiable Abstract Interpretation for Provably Robust Neural Networks. In: *Proceedings of the 35th International Conference on Machine Learning*. pp. 3578–3586. PMLR (Jul 2018)
51. Raghuathan, A., Steinhart, J., Liang, P.: Certified Defenses against Adversarial Examples. In: *International Conference on Learning Representations* (Feb 2018)
52. Roy, A., Antony, A., Gimelli, A., Daggitt, M.L.: VNN-LIB 2.0: Rigorous Foundations for Neural Network Verification (May 2026). <https://doi.org/10.48550/arXiv.2605.07451>
53. Serafini, L., d’Avila Garcez, A.: *Logic Tensor Networks: Deep Learning and Logical Reasoning from Data and Knowledge* (Jul 2016). <https://doi.org/10.48550/arXiv.1606.04422>

54. Seshia, S.A., Desai, A., Dreossi, T., Fremont, D.J., Ghosh, S., Kim, E., Shivakumar, S., Vazquez-Chanlatte, M., Yue, X.: Formal Specification for Deep Neural Networks. In: Lahiri, S.K., Wang, C. (eds.) *Automated Technology for Verification and Analysis*. pp. 20–34. *Lecture Notes in Computer Science*, Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-030-01090-4_2
55. Shi, Z., Jin, Q., Kolter, Z., Jana, S., Hsieh, C.J., Zhang, H.: Neural Network Verification with Branch-and-Bound for General Nonlinearities (May 2024). <https://doi.org/10.48550/arXiv.2405.21063>
56. Sirman, A., Conway, F., Ciupa, J., Grünbergs, G.G., Komendantskaya, E., Hoang, T.S., Rawson, M., Bruni, A., Belle, V., Williams, M.J.: Vancomycert: A certified neuro-symbolic drug delivery system (case study). In: Avni, G., Schilling, C. (eds.) *AI Verification*. pp. 331–352. Springer Nature Switzerland, Cham (2027)
57. Ślusarz, N., Komendantskaya, E., Daggett, M., Stewart, R., Stark, K.: Logic of Differentiable Logics: Towards a Uniform Semantics of DL. In: *EPiC Series in Computing*. vol. 94, pp. 473–493. EasyChair (Jun 2023). <https://doi.org/10.29007/c1nt>
58. Stoian, M.C., Giunchiglia, E.: Beyond the convexity assumption: Realistic tabular data generation under quantifier-free real linear constraints. In: *The Thirteenth International Conference on Learning Representations* (Oct 2024)
59. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks (Feb 2014). <https://doi.org/10.48550/arXiv.1312.6199>
60. Talevi, A., Bellera, C.L.: One-Compartment Pharmacokinetic Model. In: *The ADME Encyclopedia: A Comprehensive Guide on Biopharmacy and Pharmacokinetics*, pp. 1–8. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-51519-5_58-1
61. Tran, H.D., Yang, X., Manzananas Lopez, D., Musau, P., Nguyen, L.V., Xiang, W., Bak, S., Johnson, T.T.: NNV: The Neural Network Verification Tool for Deep Neural Networks and Learning-Enabled Cyber-Physical Systems. In: Lahiri, S.K., Wang, C. (eds.) *Computer Aided Verification*. pp. 3–17. *Lecture Notes in Computer Science*, Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_1
62. Tsipras, D., Santurkar, S., Engstrom, L., Turner, A., Madry, A.: Robustness May Be at Odds with Accuracy. In: *International Conference on Learning Representations* (Sep 2018)
63. Urban, C., Miné, A.: A Review of Formal Methods applied to Machine Learning (Apr 2021). <https://doi.org/10.48550/arXiv.2104.02466>
64. van Krieken, E., Acar, E., van Harmelen, F.: Analyzing Differentiable Fuzzy Logic Operators. *Artificial Intelligence* **302**, 103602 (Jan 2022). <https://doi.org/10.1016/j.artint.2021.103602>
65. van Krieken, E., Badreddine, S., Manhaeve, R., Giunchiglia, E.: ULLER: A Unified Language for Learning and Reasoning (May 2024). <https://doi.org/10.48550/arXiv.2405.00532>
66. Varnai, P., Dimarogonas, D.V.: On Robustness Metrics for Learning STL Tasks. In: *2020 American Control Conference (ACC)*. pp. 5394–5399 (Jul 2020). <https://doi.org/10.23919/ACC45564.2020.9147692>
67. Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C.J., Kolter, J.Z.: Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Complete and Incomplete Neural Network Robustness Verification (Oct 2021). <https://doi.org/10.48550/arXiv.2103.06624>
68. Wong, E., Kolter, Z.: Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope. In: *Proceedings of the 35th International Conference on Machine Learning*. pp. 5286–5295. PMLR (Jul 2018)
69. Wong, E., Schmidt, F., Metzen, J.H., Kolter, J.Z.: Scaling provable adversarial defenses. In: *Advances in Neural Information Processing Systems*. vol. 31. Curran Associates, Inc. (2018)
70. Wu, H., Isac, O., Zeljić, A., Tagomori, T., Daggett, M., Kokke, W., Refaeli, I., Amir, G., Julian, K., Bassan, S., Huang, P., Lahav, O., Wu, M., Zhang, M., Komendantskaya, E., Katz, G., Barrett, C.: Marabou 2.0: A Versatile Formal Analyzer of Neural Networks (May 2024). <https://doi.org/10.48550/arXiv.2401.14461>

71. Xu, J., Zhang, Z., Friedman, T., Liang, Y., den Broeck, G.V.: A Semantic Loss Function for Deep Learning with Symbolic Knowledge (Jun 2018). <https://doi.org/10.48550/arXiv.1711.11157>
72. Xu, K., Shi, Z., Zhang, H., Wang, Y., Chang, K.W., Huang, M., Kailkhura, B., Lin, X., Hsieh, C.J.: Automatic Perturbation Analysis for Scalable Certified Robustness and Beyond (Oct 2020). <https://doi.org/10.48550/arXiv.2002.12920>
73. Xu, K., Zhang, H., Wang, S., Wang, Y., Jana, S., Lin, X., Hsieh, C.J.: Fast and Complete: Enabling Complete Neural Network Verification with Rapid and Massively Parallel Incomplete Verifiers (Mar 2021). <https://doi.org/10.48550/arXiv.2011.13824>
74. Zhang, H., Wang, S., Xu, K., Li, L., Li, B., Jana, S., Hsieh, C.J., Kolter, J.Z.: General Cutting Planes for Bound-Propagation-Based Neural Network Verification (Dec 2022). <https://doi.org/10.48550/arXiv.2208.05740>
75. Zhang, H., Weng, T.W., Chen, P.Y., Hsieh, C.J., Daniel, L.: Efficient Neural Network Robustness Certification with General Activation Functions (Nov 2018). <https://doi.org/10.48550/arXiv.1811.00866>