Certification of avionic software based on machine learning: the case for formal monotony analysis

Mélanie Ducoffe, Christophe Gabreau, Ileana Ober, Iulian Ober, Eric Guillaume Vidot

Special Issue: FMICS 2022

Certification of avionic software based on machine learning: the case for formal monotony analysis

Mélanie Ducoffe[1] · Christophe Gabreau[1] · Ileana Ober[2] · Iulian Ober[3] · Eric Guillaume Vidot[1]

Abstract

The use of machine learning (ML) in airborne safety-critical systems requires new methods for certification, as the current standards and practices were defined and refined over decades with classical programming in mind and do not support this new development paradigm. This article provides an overview of the main challenges to the demonstration of compliance with regulation requirements raised by the use of ML and focuses on one particular case where the formal verification may become mandatory in future regulations, which is the verification of (partial) monotony properties. For this case, we propose a method to evaluate the monotony property using mixed integer linear programming. Contrary to the existing literature, our analysis provides a lower and upper bound of the space volume where the property does not hold, that we denote "Non-Monotonic Space Coverage". This work has several advantages: (i) our formulation of the monotony property works on discrete inputs, (ii) the iterative nature of our algorithm allows for refining the analysis as needed, and (iii) from an industrial point of view, the results of this evaluation are valuable for the aeronautical domain, where it can support the certification demonstration. We applied this method to an avionic case study (braking distance estimation using a neural network) where the verification of the monotony property is of paramount interest from a safety perspective.

Keywords Machine learning · Neural network · Certification · Formal verification · Monotony

1 Introduction

Over the last years, machine learning (ML) based on neural networks has become increasingly popular and a reference method for solving a broad set of problems, such as computer vision, pattern recognition, obstacle detection, time series analysis, or natural language processing. Its use in safety-critical embedded systems (e.g., automotive or aviation) is also becoming increasingly appealing, as it opens the door to new functions such as navigation/surveillance assistance (e.g. vision-based navigation, obstacle sensing, virtual sensing), autonomous flight, predictive maintenance or cockpit assistance.

The aeronautical domain is known to be one of the most stringent. Indeed, products are ruled by binding regulation requirements to guarantee that the aircraft will safely operate in foreseeable operating and environmental conditions. At the end of 2021, the European Union Aviation Safety Agency (EASA) released the first issue of a concept paper [10] to anticipate any application for AI-based products: it contains a first set of technical objectives and organization provisions that EASA anticipates as necessary for the approval of Level 1 AI applications ("assistance to human") and guidance material that could be used to comply with those objectives.

Defining and meeting the regulatory requirements for airborne software items based on ML raises a set of specific challenges, that we analyze in Sect. 2. Although work is still in progress on the side of certification bodies, it is expected

All authors contributed equally to this work.

✉ I. Ober
iulian.ober@isae-supaero.fr

M. Ducoffe
melanie.ducoffe@airbus.com

C. Gabreau
christophe.gabreau@airbus.com

I. Ober
ileana.ober@irit.fr

E.G. Vidot
eric-guillaume.vidot@airbus.com

[1] Airbus Opération S.A.S., Toulouse, France

[2] University of Toulouse, Institut de Recherche en Informatique de Toulouse, Toulouse, France

[3] Fédération ENAC ISAE-SUPAERO ONERA, Université de Toulouse, Toulouse, France

that formal verification of certain safety properties will play an important part in the assurance cases for these items.

Much of the recent work in property verification for ML-based systems was dedicated to robustness properties [3, 19, 21, 25, 29, 34, 35, 38–40, 42, 43]. However, although robustness is a critical property for classification tasks, we see the emergence of other safety-related properties for regression tasks in many systems.

Consider the case of neural network-based surrogates for approximating numerical models [2, 17, 33]. The original numerical models are developed to approximate physical phenomena, and are based on physical equations whose relevance is asserted by scientific experts. The qualification of these numerical models is carried out without any issue. However, since their computational costs and running time prevent us from embedding them on board, the use of these numerical models in the aeronautical field remains mainly limited to the development and design phase of the aircraft. ML-based surrogates alleviate this computational cost, thus opening the way for their use in airborne software. However, surrogates need to demonstrate certain safety properties stemming from the physics they model. One of them is the monotony, which is motivated by the fact that monotonic functions are ubiquitous in many physical phenomena. For instance, the braking distance of an aircraft is a monotonic function with respect to specific parameters such as the state of the brakes (nominal or degraded) or the state of the runway (dry or wet). A surrogate that estimates the braking distance should demonstrate these monotony properties. Another case where monotony is relevant is deep neural networks used for control.

Today, state-of-the-art methods for enforcing partial monotony assume that if the property is not respected on the whole operational domain of the ML-based function, this puts at risk its certification (i.e., its compliance determination to certification requirements) and therefore its industrialization. We believe that this risk can be covered, especially for models that, in the future, would also be penalized for being too loose given the reference function. In Sect. 3 we propose an iterative method that measures and identifies the part of the domain for which the monotony property is violated, which can be used to demonstrate conformity to certification requirements.

Section 4 discusses the case study and the experimental results that we have obtained with our method. We conclude the paper with a discussion of related work (Sect. 5) and our conclusions (Sect. 6).

2 Certification and machine learning

In the context of critical embedded systems, during the last few decades, the certification has established itself as a process that allows the demonstration of conformity to the regulation constraints, especially through creating and improving industrial standards that comply with the regulatory text. The trend towards obtaining (parts) of the critical systems through ML is challenging the current certification approach. To make our discussion more concrete, we refer to the certification in civil aviation. In Sect. 2.1, we present the main characteristics of the current certification approaches, and in Sect. 2.2 we analyze how the current certification practice is impacted by the emergence of systems that include parts obtained using machine learning. This analysis allows us to identify some challenges raised by the need to certify systems including parts obtained through ML, as described in Sect. 2.3. Although this analysis is made in the particular context of avionics, the analysis can be easily transposed to other contexts where certification is needed, and the challenges identified in this section correspond to issues that need to be addressed, regardless of the application domain in order to be able to certify systems including parts obtained through machine learning.

2.1 Certification practice

Nowadays, an airplane cockpit offers many complex avionic functions: flight controls, navigation, surveillance, communication, displays, etc. As a result, software plays a significant role in aircraft safety. Since the aircraft can carry up to several hundred passengers (e.g., airplanes) and their safety relies on the detection of systems failures, to which software faults can contribute, a specific development process is necessary to ensure the correctness of the embedded software. The regulatory authorities are in charge of setting up the rules that the aircraft's manufacturer and suppliers must comply with to deliver any aircraft.

The regulatory authority of the European Union – European Union Aviation Safety Agency (EASA) – provides regulatory materials in which all requirements for development of safe avionic products are defined and explained [9]. The regulatory requirements for software/hardware items embedded in avionic safety-related systems are described in Certification Specification documents (CS 2x.1301 and CS 2x.1309), out of which the CS-25 [9] is the regulatory text for large airplanes. In particular, CS-25.1301 states that "the system must perform exactly as specified in the requirements" (i.e., it performs its intended function).

Since not all the parts of the avionics system are equally critical, different levels of rigor are required within the development process depending on the failure conditions the improper functioning of the system may contribute to. Figure 1 shows the relationship between the failure severity category and its probability of occurrence. Intuitively, the more severe a failure condition is, the less probable the occurrence should be. Note that no probability is required for

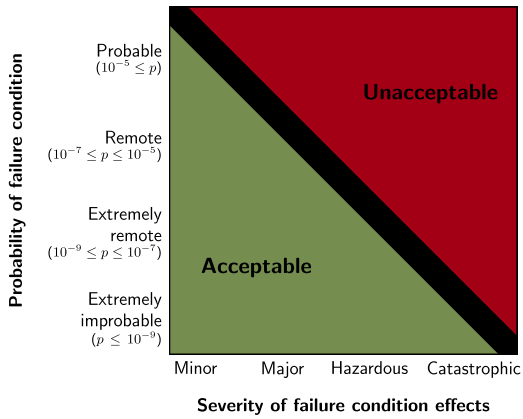

Fig. 1 Relationship between probability and severity of failure condition effects (adapted from [9, Fig. 1 AMC 20.1309, p.779])

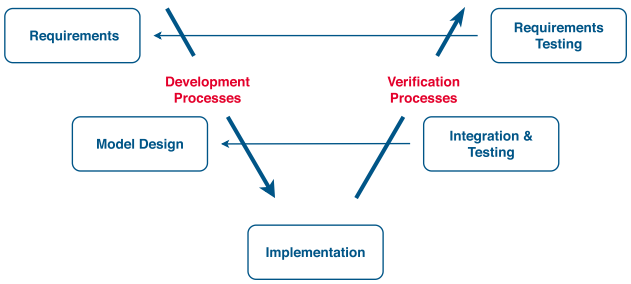

Fig. 2 Item development workflow

the failure conditions with *no safety effect*. On the contrary, a *catastrophic* failure condition must be at most *extremely improbable*.

2.1.1 Software certification

Contrary to the hardware assessment, no metric exists to measure the software quality. Based on the observation that the development process's quality directly impacts the software's quality, the standard DO-178x is defined as a set of software considerations in airborne systems. It defines requirements in terms of objectives (e.g., source code complies with low-level requirements), activities (e.g., code review) and evidence (e.g., document summarizing the code review results).

In the context of the DO-178C standard, the complete standardized workflow comprises the planning process, four development processes (requirement, design, coding, integration), and four integral processes (verification, configuration management, quality assurance, and certification liaison). In the context of this article, only the development and verification processes are meaningful.

In the context of the standard DO-178, the item's development workflow is represented by the V-cycle, summarized in Fig. 2. The usual development paradigm is the requirement-based engineering (either textual or model-based): it corresponds to the activities link to the top left square of Fig. 2. This step might corresponds to the definition of the *high-level requirements,* which are refined during the development process leading to the software architecture and the definition of the *low-level requirements*.

Once high-level requirements, low-level requirements, and the software architecture are well defined, the implementation can start (bottom of the V-cycle). For the sake of the complete description of the software behavior, the specification can be completed with derived requirements, i.e., non-traceable requirements to higher-level requirements. In parallel, specific verification activities interfere with each development activity.

To summarize, each line of code implemented in a software must be

- developed and traceable from the low-level requirements
- reviewed against the low-level requirements
- covered by requirement-based tests (either high-level requirements and/or low-level requirements)

Moreover, several activities (e.g., review or verification activities) may need to be performed independently depending on the targeted assurance level.

2.2 Impact of ML on certification approaches

The introduction of parts of code obtained using ML challenges the current certification practices, as the existing standards do not provide sufficient guidance to make a complete demonstration of conformity for an ML-based system. Indeed, some fundamentals of the usual techniques (in particular related to the pivotal place of the requirement-based engineering and model-based engineering) are jeopardized, challenging the classical safety guarantee argumentation.

Let us note here that since not all the software used in avionics needs to be certified, it is possible even with nowadays certification standards to integrate ML into parts rated at Design Assurance Level (DAL) "E". For an analysis and classification of AI applications and their compatibility with current certification standards, the reader is referred to Schweiger et al [31].

For parts of code obtained using ML that require certification, three gaps induced by the use of ML can be directly identified: the lack of specifiability, the lack of traceability, and the potential introduction of unintended behavior that traditional methods cannot handle. These gaps are quickly described below, readers interested in a deeper analysis of these challenges of certifying ML-based items can refer to Mamalet et al [22].

2.2.1 Specifiability

As we have seen in the previous section, the current certification practice relies heavily on the requirement specification. ML-based items are often used when it is impossible or very difficult to specify the function entirely with a classical requirement process. For instance, the runway detection specification in the aeronautical industry should contain all the possible ways to describe a runway, whatever the environmental and operational conditions (e.g., weather, light conditions, sensor bias). This cannot be achieved using textual requirements or a modeling technique. For this reason, a ML-based approach is preferred in the presence of a training dataset of thousands to millions of images whose role would be to complete the requirements capture and allow the development of an acceptable runway detection function.

One can argue that the absence of a complete specification inherently decreases the confidence that the model behavior always matches the functional intent and, therefore, is free of unintended behavior that may jeopardize safety. One strategy to mitigate this problem consists of ensuring the completeness and the quality of the dataset. For example, for a runway detection function, the idea is to describe as thoroughly as necessary the Operational Design Domain (ODD) in which the detection function is supposed to operate and then validate that the collected data correctly and sufficiently represents this ODD. Another mitigation technique, consists in formally specifying safety properties that the ML model should satisfy for any combination of inputs. For example, for a braking distance estimation function constructed by ML, the idea is to formally specify specific properties, such as bounded variation or monotony, with respect to certain parameters (weight, speed). The satisfaction of such properties of the ML model can, in some cases, be verified, e.g., using an SMT or MILP solver, as we will see in Sect. 3.

2.2.2 Traceability

The current standard for certified software item development — DO-178C — requires traceability between the requirements and the code (in both senses) so that each line of embedded code can be traced to the requirements that lead to it and then justified as implementing captured requirements. In an ML development context, this relationship, which makes the design a white-box process, is lost. The trained algorithm is a complex parametric mathematical expression where the values of the learned parameters are not traceable to any upward functional requirement. Thus it becomes infeasible to demonstrate the completeness and the correctness of implementation using traceability.

This lack of traceability results in the loss of transparency of the model, making the link from the input data to the output predictions not understandable by humans. The traceability loss lowers the confidence that the safety properties of the intended function are preserved in its operational domain. In this context, explainable AI is seen as a means to enhance confidence in these algorithms when safety is highly critical.

2.2.3 Unintended behavior

The entire life-cycle processes (requirements capture, model design, implementation, integration, and requirement-based testing) of the item development workflow (see Fig. 2) is used to demonstrate the consistency between the obtained implementation and the specified requirements. In addition to the problems mentioned above, the learning phase could introduce some unexpected behavior, such as high sensitivity to perturbations, that cannot be measured or prevented by usual verification and validation activities. Consequently, one cannot guarantee the absence of unintended behavior during item operation, specifically those affecting aircraft safety.

Unintended behavior may result from various causes, such as a low-quality data management process (e.g., biased or mislabeled data) or an inadequate learning process. Some theoretical results emerge that allow to limit or formally verify the absence of unintended behavior.

2.3 Challenges of certifying systems containing ML

The certification of ML-based items in the framework of avionic developments opens challenging (research) domains, as illustrated in Fig. 3.

We present these challenges through the lens of the avionic domain with the perspective of conformity to the regulation requirements. In 2021, the EASA released its first usable guidance regarding the certification of machine learning applications [10]: this document gathers the first set of objectives in order to anticipate future EASA guidance and requirements and frames any future development of an ML-based function as a part of a safety-critical system.

Due to the large number of domains involved in the qualification of an ML-based item, we focus on the software level of engineering. As a result, system-level embedding and resilience issues, such as fault tolerance, are not addressed here. We spotlight five domains that should contribute significantly to support the qualification of ML-based items: explainability, provability, adversarial robustness, data, and methodology (see Fig. 3).

Explainability aims to help human decision makers understand the behaviour and outcomes of ML items. One can expect that, when required by the safety level of the implemented function, explainability is requested to add the necessary confidence to support the demonstration of conformity [10].

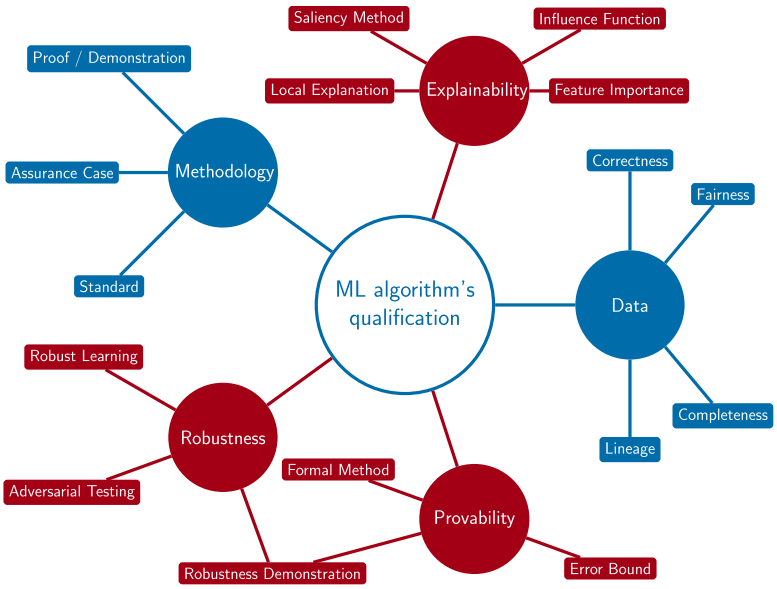

Fig. 3 Key domains involved in ML certification

Phillips et al [27] identifies four principles of explainable AI: explanation, meaningfulness, explanation accuracy, and knowledge limits. These principles state that an explainable AI must: *(i)* provide an "evidence or reason for all outputs", *(ii)* be meaningful with respect to the audience, *(iii)* be representative of the way that the algorithm produces the output, and *(iv)* be aware of the domain of usage of the algorithm, i.e., it should not give an explanation when the input is out of scope, as the algorithm is not designed to work with it [23].

In the avionic context, four stakeholders could need explanation: the designers, the authorities, the end users (e.g., pilots), and the forensic investigators. As one can notice, the nature of explanation may vary depending on the stakeholder. For instance, considering the "meaningfulness" principle, an engineer who knows the system would need a different explanation than an end-user who has no prior or inner knowledge of the system. The explanations could also differ in their nature whether it is used for debugging purposes during the development (for the designer), for investigation purposes in case of in-flight issues (for authorities or investigator), or for a user assessment of the model predictions when the system is deployed, e.g., a pilot who requests justification of the decision to enhance his confidence in the system before acting. As far as the ML items certification is concerned, formal verification and formal explanations [23] will offer essential guarantees of rigor.

Robustness comprises several sub-domains (adversarial examples, distributional shift, unknown classes, physical attacks, etc.). We focus here on adversarial robustness, which is the capability of algorithms to give the same outputs considering some variation of the inputs in a region of the state space. The issues addressed by the adversarial robustness domain are at the heart of the software qualification process, as it partially addresses the demonstration that there are no unintended functions. The aim of the robustness is to assess and enhance the algorithm's behavior when dealing with perturbed inputs (e.g., noise, corner cases, or sensor malfunction). One way to improve the adversarial robustness is finding robust learning procedures that are resilient against crafted examples made to defeat the ML algorithm.

Data management refers to the practice of collecting, organizing, and accessing data. In the avionic context, data has been used for a long time, with systems making heavy use of databases or configuration files. In the context of ML, the design techniques are data-driven, thus ML makes the data management process essential to the demonstration of conformity. Building a good ML algorithm requires good quality data (e.g., no erroneous or mislabeled data), yet this is not sufficient, as data representativeness also plays a significant role. Representativeness refers to checking if the data is an accurate snapshot of the phenomenon to be learned.

The quality of the data can be measured using metrics [28] such as: accuracy (data is well measured and stored), consistency (relationship between parameters regarding the operational design domain), integrity (not corrupted data), timeliness (data availability and correctness over time), traceability (data source reliability), and fairness (lacking undesirable biases).

Methodology structures the assurance activities needed to support the qualification aspects, i.e., the demonstration to the authorities that the item development complies with the recognized and standardized guidance.

It will be crucial for the future Acceptable Means of Compliance (AMC) and/or the industrial standards to clearly de-

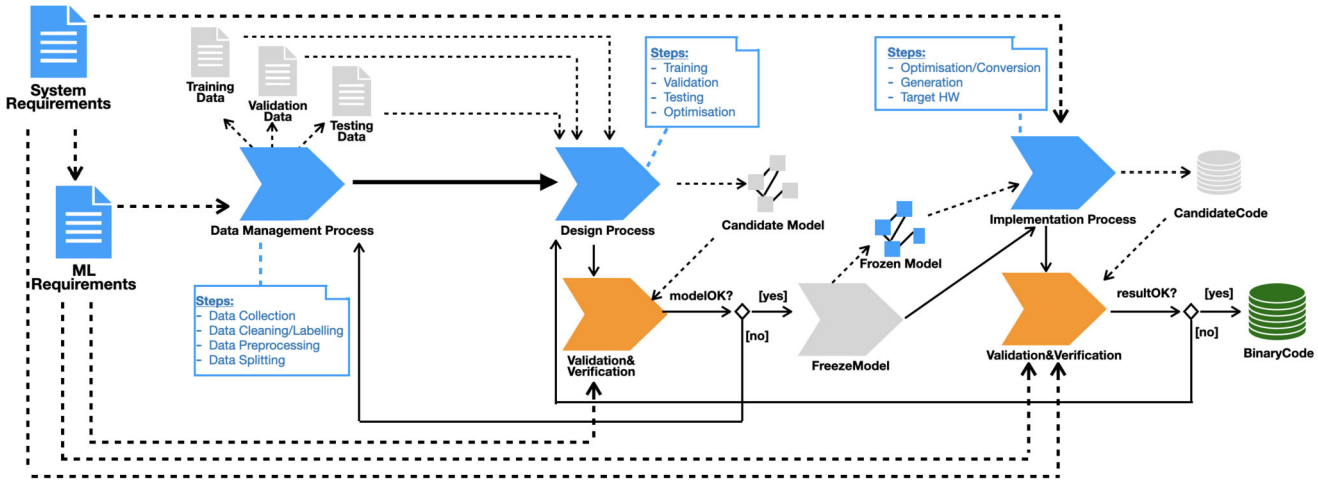

Fig. 4 ML items development process

fine the development process of ML-based items, starting from existing best practices [1], in order to *(i)* ease their development and maintainability, *(ii)* identify the validation and verification activities, and *(iii)* guarantee their certifiability. Figure 4 describes this development process: requirements from the system/subsystem level are refined into ML-based item requirements to fit the three main stages of the workflow:

- Data management process covers data collection, data cleaning and labeling, data pre-processing – comprising various steps used to transform the data into a more suitable format for the design phases, e.g., feature engineering or data normalization, and data splitting into a training, a validation, and a testing dataset.
- Design process takes as input the three datasets, it outputs a frozen ML model (which means no retraining afterward). The processes in this stage are iterative rather than sequential. Indeed, iteration between the training and the validation steps of the design process is done in order to tune the hyper-parameters of a model. After this tuning, the model is tested, leading to a candidate model. The candidate model is subject to validation with respect to the ML item requirements. If the performances are lower than expected, it is possible to loop back to the data management process to improve the model. The model is frozen when it attains the performance criteria set before implementation.
- Implementation process takes as an input the frozen model obtained through ML and it optimizes it with respect to the host platform constraints and the operational requirements cascaded from the system level. Finally, a binary code is generated and loaded into the hardware target.

Validation and verification activities exist at the various stages to ensure the quality of the respective outputs.

Since their creation in the 1980s, the DO-178x standards gather the industrial best practices imposing development rigor needed to avoid introducing errors that may lead to a system failure, and thus increasing the overall safety of the system. Historically, methodologies undeniably brought efficient results in terms of safety. Thus methodological considerations are key to build a viable certification approach. An essential element in building relevant safety argumentation are the non-prescriptive approaches, such as safety cases [30] or assurance cases [24]. Assurance cases, which are a generalization of safety cases, are defined by MITRE [24] as follows: *a documented body of evidence that provides a convincing and valid argument that a specified set of critical claims regarding a system's properties are adequately justified for a given application in a given environment*. Hence, assurance cases could be used during the development process of an ML-based item to build, share and discuss sets of structured arguments to support the demonstration of conformity based on outcomes of specific assurance techniques [7].

The development process for software items using ML, sketched in Fig. 4, is what we consider today to be the most likely one to be used in certification in the future. Alternative approaches for including ML-based components in certified software (e.g., [6] – in the context of avionics or [4] – in the context of automotive) exist and rely on the use of a ML-based component that uses run time monitoring to anticipate critical situations that could lead to using a ML-free certified backup component.

Provability is the capability to formally demonstrate that model properties are preserved. Formal methods provide mathematical evidence to support such a demonstration. Since robustness, as presented above, can be expressed as a property, the provability covers the adversarial robustness

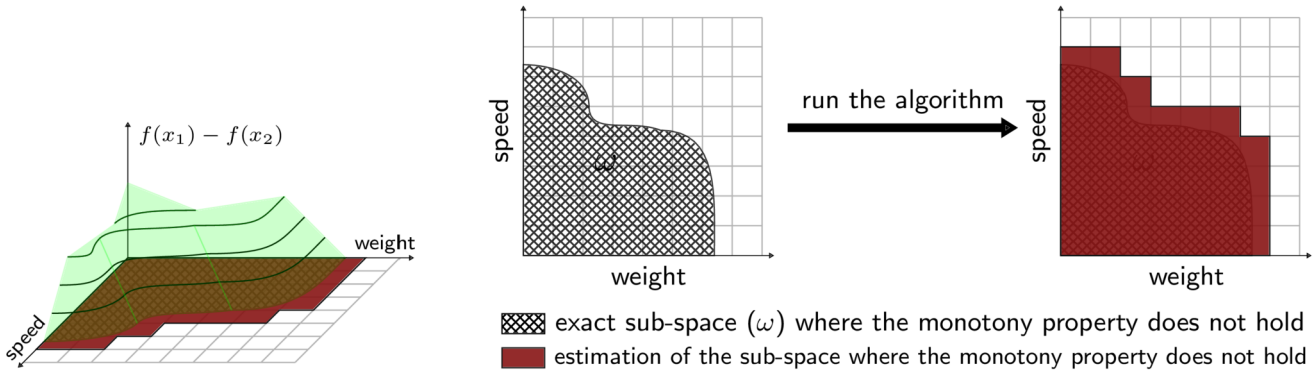

Fig. 5 Purpose of the algorithm through Example 1. In x_1 the runway is dry and in x_2 the runway is wet. The left plot represents $f(x_1) - f(x_2)$ where only the positive values are displayed (monotony property violated). The two plots on the right are the projection of these points on the plane composed of features 1 and 2. (Color figure online)

demonstration. Indeed, formal methods are already used to verify well-defined properties of models, such as the robustness [32] but also safety properties [19]. Regarding the aeronautical context, verifying models' properties, either functional or safety-related (like adversarial robustness), is an asset for its qualification, since it may avoid time and cost-consuming testing while providing formal proof that the property holds.

In the rest of this paper, we focus on one particular case where the formal verification may become mandatory in future regulations, which is the verification of (partial) monotony properties. With respect to the key domains related to the challenges on certifying systems containing ML illustrated in Fig. 3, the formal verification will contribute to the domains depicted in red, namely robustness, provability and explainability.

3 Monotony analysis

In this section, we define the concept of *partial* monotony with respect to a set of inputs. Let $\mathcal{V}$ be a (finite) set of input features. For each feature $v \in \mathcal{V}$ we denote $\mathcal{D}(v)$ the domain in which v ranges. Hence, let $X = \times_{v \in \mathcal{V}} \mathcal{D}(v)$ be the input space, Y be the output space and $f : X \to Y$ be the neural network. Note that the features are generally of two types ($\mathcal{V} = \mathcal{V}_d \sqcup \mathcal{V}_c$):

- $v \in \mathcal{V}_d$ are features whose domain $\mathcal{D}(v)$ is discrete (e.g., a finite set of labels or categorical values)
- $v \in \mathcal{V}_c$ are features whose domain $\mathcal{D}(v)$ is a real interval

In this work, we are interested in monotony properties, which supposes that the set Y has an order relation denoted $\preceq$; usually, $Y \subseteq \mathbb{R}$ and $\preceq$ is one of the usual orders ($\leq$, $\geq$). The monotony property will be relative to a subset of discrete features, $\alpha \subseteq \mathcal{V}_d$ for which a partial order is defined on $\times_{v \in \alpha} \mathcal{D}(v)$, also denoted $\preceq$ without risk of confusion. We denote with $\prec$ the strict order derived from $\preceq$, i.e., $s \prec t$ iff $s \preceq t$ and $s \neq t$. For $x \in X$, let us denote $x{\downarrow_\alpha}$ the projection of x onto the dimensions in α, and $\bar{\alpha} = \mathcal{V} \setminus \alpha$.

Definition 1
Monotony Property A function f is monotone with respect to an order $\preceq$ on the output domain Y and to a subset of discrete features $\alpha \subseteq \mathcal{V}_d$ endowed with a partial order defined on $\times_{v \in \alpha} \mathcal{D}(v)$ also denoted $\preceq$ (without risk of confusion), if and only if

$$\forall (x_1, x_2) \in X^2 : x_1{\downarrow_{\bar{\alpha}}} = x_2{\downarrow_{\bar{\alpha}}} \ \wedge \ x_1{\downarrow_\alpha} \preceq x_2{\downarrow_\alpha}$$
$$\implies \ f(x_1) \preceq f(x_2)$$

3.1 Goal of the analysis

Our analysis aims to identify the sub-spaces where the monotony does not hold using a MILP solver. Example 1 describes a toy example (a simplified version of the case study in Sect. 4) that we will use to explain the main concepts.

Example 1
Setup: Let f be a neural network estimating the braking distance of an aircraft based on its speed, its weight and the runway's state (dry or wet).

Property: *for the same speed and weight ($x_1{\downarrow_{\bar{\alpha}}} = x_2{\downarrow_{\bar{\alpha}}}$), the braking distance on a wet runway must be higher than on a dry one ($x_1{\downarrow_\alpha} \preceq x_2{\downarrow_\alpha} \implies f(x_1) \preceq f(x_2)$).* Goal: Identify and quantify the input areas where the property does not hold.

If we plot $f(x_1) - f(x_2)$ versus the speed and the weight, the Definition 1 holds if and only if all the values are negative or null. The 3D plot in Fig. 5 shows a sketch of this example when the monotony property partially holds, i.e., $f(x_1) - f(x_2)$ is partially positive. To ease the visualization

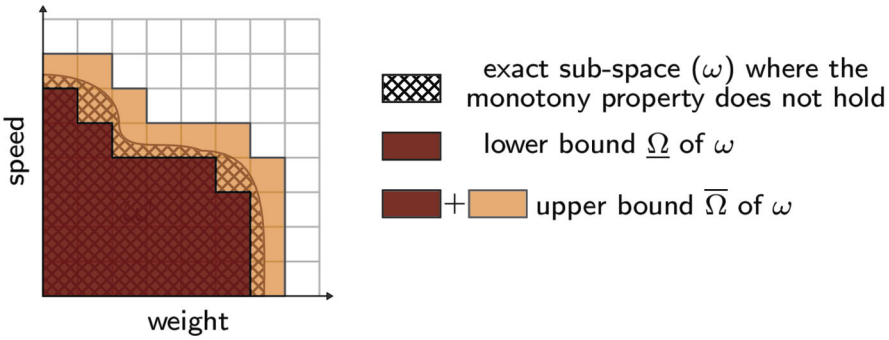

Fig. 6 Based on Fig. 5: representation of $\underline{\Omega}$ and $\overline{\Omega}$ considering Example 1 (Color figure online)

we only draw the positive values. The crosshatched area in the 2D plots are projections of the positive values of the curve on the plane representing the speed and the weight features and models the area where the monotony property is not respected, namely the non-monotonic space coverage, denoted as ω. The rightmost 2D plot shows what we expect from our analysis on Example 1: identifying and estimating ω. To estimate ω, we partition the space (grid in Fig. 5) and then the monotony property is checked on each sub-space. The dark red area represents the identified sub-space where monotony issues occur, i.e., an over-approximation of ω. In addition, our approach provides a lower and upper bound of the size of ω relative to the whole input domain, respectively denoted as $\underline{\Omega}$ and $\overline{\Omega}$ (See Fig. 6).

Our approach can distinguish the sub-spaces where the monotony property does not hold (dark red area in Fig. 6) from the ones where it partially holds (orange area in Fig. 6). Hence, the lower bound is the dark red area, while the upper bound is the dark red and orange areas. The benefit of having a lower and upper bound, instead of just an overestimation, is to be able to assess whether our estimation is precise: large gaps between the upper and lower bounds may reveal that our bounds are not representative of ω. The iterative nature of our approach overcomes this problem: we refine our space, which leads to a finer grid for the Fig. 6, and run again the MILP solver where the property partially holds to have a most accurate estimation of ω.

3.2 MILP formulation

3.2.1 Neural network encoding

Let $f : X \rightarrow Y$ be a neural network composed of n layers with ReLU activations. The layer 0 corresponds to the input layer while the layer n to the output one. We use the MILP formulation proposed by Cheng et al [5], which uses the big-M method [13] to encode the ReLU activation. By convention, the notations in bold denote the MILP variables, and those not in bold denote constants. For $1 \leq i \leq (n-1)$, let C^i be the conjunction of constraints for the layer i:

$$C^i \triangleq \quad \hat{\mathbf{x}}^{\mathbf{i}} = W^i \mathbf{x}^{\mathbf{i-1}} + b^i \tag{1}$$

$$\wedge\ \mathbf{x}^{\mathbf{i}} \leq \hat{\mathbf{x}}^{\mathbf{i}} + M^i(1 - \mathbf{a}^{\mathbf{i}}) \quad \wedge \quad \mathbf{x}^{\mathbf{i}} \geq \hat{\mathbf{x}}^{\mathbf{i}} \tag{2}$$

$$\wedge\ \mathbf{x}^{\mathbf{i}} \leq M^i \cdot \mathbf{a}^{\mathbf{i}} \quad \wedge \quad \mathbf{x}^{\mathbf{i}} \geq 0 \tag{3}$$

$$\wedge\ \mathbf{a}^{\mathbf{i}} \in \{0,1\}^{|\mathbf{x}^{\mathbf{i}}|}, \tag{4}$$

where $\hat{\mathbf{x}}^{\mathbf{i}}$ and $\mathbf{x}^{\mathbf{i}}$ are the vector of neuron values at the layer i before and after the ReLU activation, respectively. M^i is a large valid upper bound s.t. $-M^i \leq \hat{\mathbf{x}}^{\mathbf{i}}$ and $\mathbf{x}^{\mathbf{i}} \leq M^i$ [5]. W_i and b_i are, respectively, the weights and bias at the layer i, and $\mathbf{a}^{\mathbf{i}}$ is a binary vector that represents whether the neurons are activated or not. Equation (1) is the constraint for the affine transformation and the Equations (2)-(4) are the constraints encoding the ReLU activation. For the output layer n, there is no ReLU activation, then we have:

$$C^n \triangleq \quad \hat{\mathbf{x}}^{\mathbf{n}} = W^n \mathbf{x}^{\mathbf{n-1}} + b^n \tag{5}$$

It remains to encode the constraints of the input layer, which enforce the lower and upper bounds of the domain of the input features. Our analysis relies on a partition of the input space X, thus the encoding of the input layer depends on it: let $\mathcal{P}$ be a partition of X, $p \in \mathcal{P}$ be a subset of X represented by a set of linear constraints (also denoted p). Hence, the neural network f is encoded as the conjunction of the constraints defined for each layer and p, which is constraining the input layer:

$$C^f(p) \triangleq p \wedge \left(\bigwedge_{i=1}^{n} C^i \right) \wedge C^n \tag{6}$$

3.2.2 Monotony property encoding

Following Definition 1, we must encode f twice in MILP: C_1^f and C_2^f. Similarly to the encoding of the input space's constraints, we encode the monotony property regarding the partition $\mathcal{P}$. So, let $p_i, p_j \in \mathcal{P}^2$ be two sub-spaces of X such that $\exists x_1, x_2 \in p_i \times p_j$, $x_1{\downarrow_{\bar{\alpha}}} = x_2{\downarrow_{\bar{\alpha}}} \wedge\ x_1 \prec x_2$. Then, we have:

$$C^{mon}(p_i, p_j) \triangleq \left(\mathbf{x_1^0}{\downarrow_{\bar{\alpha}}} = \mathbf{x_2^0}{\downarrow_{\bar{\alpha}}} \wedge\ \mathbf{x_1^0}{\downarrow_{\alpha}} \preceq \mathbf{x_2^0}{\downarrow_{\alpha}} \right) \wedge$$

$$\left(C_1^f(p_i) \wedge C_2^f(p_j)\right) \wedge \left(\hat{\mathbf{x}}_1^{\mathbf{n}} \leq \hat{\mathbf{x}}_2^{\mathbf{n}}\right) \quad (7)$$

$$C^{\neg mon}(p_i, p_j) \triangleq \left(\mathbf{x_1^0}{\downarrow_{\tilde{\alpha}}} = \mathbf{x_2^0}{\downarrow_{\tilde{\alpha}}} \wedge\ \mathbf{x_1^0}{\downarrow_{\alpha}} \preceq \mathbf{x_2^0}{\downarrow_{\alpha}}\right) \wedge$$
$$\left(C_1^f(p_i) \wedge C_2^f(p_j)\right) \wedge \left(\hat{\mathbf{x}}_1^{\mathbf{n}} \geq \hat{\mathbf{x}}_2^{\mathbf{n}} + \epsilon\right) \quad (8)$$

The MILP solver may output either `SAT`, `UNSAT` or `TIMEOUT`. For Equation (7) and Equation (8), `TIMEOUT` means that the time limit has been reached. C^{mon} checks whether the neural network f is monotonic:

- `SAT`: there is an assignment for $\mathbf{x_1^0}, \mathbf{x_2^0} \in p_i \times p_j$ that respects the monotony.
- `UNSAT`: the monotony is violated on the entire sub-space $p_i \times p_j$.

$C^{\neg mon}$ checks whether the neural network is not monotonic:

- `SAT`: there is an assignment for $\mathbf{x_1^0}, \mathbf{x_2^0} \in p_i \times p_j$ that violates the monotony.
- `UNSAT`: the monotony is respected on the complete sub-space $p_i \times p_j$.

Since strict inequalities cannot be used in a MILP model, we introduce the ϵ term in Equation (8) so that $C^{\neg mon}$ is not satisfied when $\hat{\mathbf{x}}_1^{\mathbf{n}} = \hat{\mathbf{x}}_2^{\mathbf{n}}$.

To determine for each sub-space $p_i \times p_j$ whether the monotony property holds, partially holds, or does not hold (see Fig. 6), we must solve successively $C^{\neg \text{mon}}$ and C^{mon} (see Sect. 3.3 for more details).

3.2.3 Note on alternative formulations

It should be noted that the network and property can be encoded in a very similar way in SAT modulo linear arithmetic, and verified with an SMT solver. We have performed this experiment using the Z3 solver [8], and the resolution proved to be several orders of magnitude slower than with MILP. Finding the reason for this difference is beyond the scope of this paper, but we believe it is related to how the SMT solver handles the large set of real parameters in W and b. Since the difference in performance makes verification with standard SMT solvers impractical, we do not report further on it.

Another possibility that can be considered is the use of a specialized theory/solver for ReLU networks such as Marabou [19]. However, our study targets networks with discrete inputs and with specific constraints (see Sect. 4), which to the best of our knowledge cannot be captured by Marabou, making its use impossible in this case.

3.3 Verification procedure

As explained in Sect. 3.1, our verification procedure implies the partition of the space and the verification of each sub-space. In Algorithm 1, the monotony property is iteratively analyzed regarding a partition, refining this partition in the zone of interest to sharpen the analysis. Algorithm 2 details the verification run at each iteration.

Algorithm 1 Monotony analysis refinement

Require: T: the number of iteration of the procedure

$P_1 \leftarrow \{(p_i, p_j) \in \mathcal{P}^2 \mid \exists (x_1, x_2) \in p_i \times p_j, x_1 \prec x_2 \text{ and } x_1{\downarrow_{\tilde{\alpha}}} = x_2{\downarrow_{\tilde{\alpha}}}\}$
$\widehat{P}_0 \leftarrow P_1$ and $\underline{\Omega_0} \leftarrow 0$
$\mathbb{P}^{\neg \text{mon}} \leftarrow \emptyset$, and $\mathbb{P}^{\text{partially mon}} \leftarrow \emptyset$
for t from 1 to T **do**
  $\widehat{P}_t \leftarrow \widehat{P}_{t-1} \wedge P_t$
  $(P_t^{\neg \text{mon}}, P_t^{\text{partially mon}}) \leftarrow \mathbf{F}(\widehat{P}_t)$
  $\underline{\Omega_t} \leftarrow \underline{\Omega_{t-1}} + \frac{|P_t^{\neg \text{mon}}|}{|P_t|}$ and $\overline{\Omega_t} \leftarrow \underline{\Omega_t} + \frac{|P_t^{\text{partially mon}}|}{|P_t|}$ {See Fig. 7}
  $\mathbb{P}^{\neg \text{mon}} \leftarrow \mathbb{P}^{\neg \text{mon}} \cup P_t^{\neg \text{mon}}$ and $\mathbb{P}^{\text{partially mon}} \leftarrow P_t^{\text{partially mon}}$
  $\widehat{P}_t \leftarrow P_t^{\text{partially mon}}$
  $P_{t+1} \leftarrow \texttt{partition}(P_t)$
end for
return $\underline{\Omega_T}$, $\overline{\Omega_T}$, $\mathbb{P}_{\neg \text{mon}}$ and $\mathbb{P}_{\text{partially mon}}$

Algorithm 2 $\mathbf{F}(P) \longrightarrow$ Monotony analysis of $P \subseteq \mathcal{P}^2$

Require: $P \subseteq \mathcal{P}^2$ gathers the sub-spaces that need to be verified.

$P^{\neg \text{mon}} \leftarrow \emptyset$
$P^{\text{partially mon}} \leftarrow \emptyset$
for all $(p_i, p_j) \in P$ **do**
  if `solve`$(C^{\neg mon}(p_i, p_j))$ is `SAT` **then**
    if `solve`$(C^{mon}(p_i, p_j))$ is `SAT` **then**
      $P^{\text{partially mon}} \leftarrow P^{\text{partially mon}} \cup \{(p_i, p_j)\}$
    else
      $P^{\neg \text{mon}} \leftarrow P^{\neg \text{mon}} \cup \{(p_i, p_j)\}$
    end if
  else
    Continue to the next (p_i, p_j) {Monotonic on the whole domain $p_i \times p_j$}
  end if
end for
return $P^{\neg \text{mon}}$ and $P^{\text{partially mon}}$ $\{P^{\neg \text{mon}} \cup P^{\text{partially mon}} \subseteq P\}$

3.3.1 Algorithm 1

The monotony is defined on the space X^2; however, we define earlier the partition $\mathcal{P}$ of X. Hence to verify the monotony on the complete space, i.e., X^2, we need to go through all the sub-spaces i.e., $p_i \times p_j, \forall (p_i, p_j) \in \mathcal{P}^2$. However, it may happen that the monotony does not apply to the sub-space (p_i, p_j) because there are no comparable elements within the

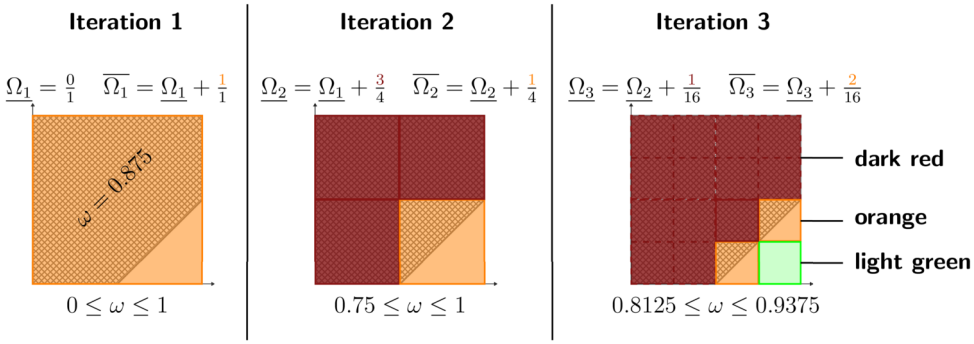

Fig. 7 Run of Algorithm 1 on Example 1 with the detailed computation of $\underline{\Omega_t}$ and $\overline{\Omega_t}$. The crosshatched area represents the sub-space the algorithm strives to estimate. (Color figure online)

Table 1 State of the monotony property regarding the condition of Lines 4 and 5

Case	Line 4	$C^{\neg mon}$	Line 5	C^{mon}	Monotony property on $p_i \times p_j$
1	True	`SAT`	True	`SAT`	partially holds
2	True	`SAT`	False	`UNSAT`	does not hold
3	False	`UNSAT`	–	–	holds

sub-space: P_1, in Line 1, contains all and only the (p_i, p_j) including comparable elements. We denote the elements of P_1 and more generally, P_t, "monotony scenario".

We propose an iterative procedure where at each iteration we use, in Line 6, $\mathbf{F}(\cdot)$ (see Algorithm 2) to retrieve $P_t^{\neg\text{mon}}$ and $P_t^{\text{partially mon}}$. Then, we compute in Line 7 the metrics $\underline{\Omega_t}$ and $\overline{\Omega_t}$ for the iteration t, which respectively lower-bounds and upper-bounds ω; ω is the exact ratio of the space where f is non-monotonic, which corresponds to the ratio of monotony scenarios where f is non-monotonic. In Line 11, we refine the partition of the space for the next iteration: `partition` is the function that takes the current partition of the space and returns a finer partition; we suppose that all elements in the partition have the same size. Note that P_t gets finer and finer through the iterations: the more we refine, the more elements P_t will have. We highlight that in Line 5, the operator $\wedge$ applies the intersection between each subset of $\widehat{P}_{t-1}$ and P_t where P_t is a finer partition of the space than $\widehat{P}_{t-1}$. It allows to get the elements of interest ($\widehat{P}_{t-1}$) in the right level of details (P_t). For the first iteration, we run $\mathbf{F}(\cdot)$ on all the elements (initialization of $\widehat{P}_0$ to P). However, we only need to refine the sub-spaces where the monotony property is partially respected for the other iterations. Finally, the algorithm returns the lower and upper bounds of each iteration and all the sub-spaces where the monotony property does not hold or partially holds.

In Fig. 7, we run Algorithm 1 on Example 1:[1] α contains the runway's state, we partition X on α and we have a unique (p_i, p_j) in P_1; in p_i the runway is dry and in p_j wet. Then, the two axes represent features of $\bar{\alpha}$ (speed and weight) and the squares, the partition of the space. The crosshatched surface is the exact sub-space where the monotony property does not hold. The orange squares mean that the monotony property partially holds, the dark red squares means it does not hold, and the light green squares means it holds. Through the iteration, we refine the partition (smaller squares), while running the verification only for the smaller squares (in solid lines) coming from a bigger orange square (in Line 5; $\widehat{P}_{t-1}$ is the orange square of iteration 2 and P_t is the small squares of iteration 3).

[1] Note that we simplify the crosshatched area's shape in order to know the omega value for the explanation.

3.3.2 Algorithm 2

The verification function $\mathbf{F}(P)$ aims to analyze the monotony of f regarding P a subset of $\mathcal{P}^2$, which gathers the sub-spaces where the monotony property must be checked. Intuitively, the partition $\mathcal{P}$ and thus P can be seen as the level of details of the monotony analysis. Indeed, a finer partition $\mathcal{P}$ results in smaller sub-spaces in P; hence a more detailed analysis.

Then, from Lines 4 to 12, we identify in which sub-spaces $p_i \times p_j$ the neural network f partially respects or does not respect the monotony property and sort them in $P^{\text{partially mon}}$ and $P^{\neg\text{mon}}$. In Lines 4 and 5, `solve(·)` refers to any off-the-shelf MILP solver taking as input a MILP problem. Table 1 shows the interpretation of the monotony of f within the sub-space regarding every truth values of the conditions of Lines 4 and 5. Note that we arrive in Line 11 when the condition of Line 4 is False, and we jump to the next sub-space (or monotony scenario) because the monotony property holds for the current sub-space $p_i \times p_j$. Finally, we return the two sets gathering the sub-spaces where the monotony property does not hold and where it partially holds.

3.3.3 Non-monotonic space coverage

$\underline{\Omega_t}$ and $\overline{\Omega_t}$ are defined as the ratio of sub-spaces (monotony scenarios) where f has monotony issue over the total number sub-spaces in P_t (contains all the monotony scenarios):

Definition 2
Lower and upper bounds of ω

$$\underline{\Omega_t} = \underline{\Omega_{t-1}} + \frac{\left|P_t^{\neg\text{mon}}\right|}{|P_t|} \tag{9}$$

$$\overline{\Omega_t} = \underline{\Omega_t} + \frac{\left|P_t^{\text{partially mon}}\right|}{|P_t|} \tag{10}$$

On the one hand, $\underline{\Omega_t}$ takes into account only the sub-spaces where the monotony property holds not; hence, it lower-bounds ω. On the other hand, $\overline{\Omega_t}$ considers the sub-spaces where the monotony property holds not and partially holds; hence, it upper-bounds ω. Figure 7 details the computation of these metrics along with the iteration: at each iteration, the lower bound $\underline{\Omega_t}$ is represented by all the dark red squares and the upper bound $\overline{\Omega_t}$ by all the dark red and orange squares.

Example 2
Computation of $\underline{\Omega_t}$ and $\overline{\Omega_t}$ considering Example 1.

Iteration 1 We consider the entire space. Hence, we only have one sub-space where we assess the monotony property ($|P_t| = 1$). There is no dark red square, i.e., sub-space where the monotony property does not hold, which means that $\left|P_1^{\neg\text{mon}}\right| = 0$, then $\underline{\Omega_1} = 0$. We have one orange square: in this sub-space, the monotony property partially holds, then $|P_1^{\text{partially mon}}| = 1$ and $\overline{\Omega_1} = 1$.

Iteration 2 We partition the space in 4 smaller sub-spaces ($|P_t| = 4$) and run again the verification on each sub-space. We proceed similarly as previously for the computation of $\underline{\Omega_2}$ and $\overline{\Omega_2}$. We have 3 dark red squares ($\left|P_2^{\neg\text{mon}}\right| = 3$) and 1 orange square ($|P_2^{\text{partially mon}}|=1$): $\underline{\Omega_2}=\underline{\Omega_1} + \frac{3}{4}=0.75$ and $\overline{\Omega_2} = \underline{\Omega_2} + \frac{1}{4} = 1$.

Iteration 3 We refine the partition of the previous step, and we end up with 16 sub-spaces. However, we only run the verification on the sub-spaces coming from an orange square (Lines 5 of Algorithm 1), i.e., a sub-spaces where f is partially monotonic. We have $\underline{\Omega_3} = \frac{3}{4} + \frac{1}{16} = 0.8125$ and $\overline{\Omega_3} = \frac{3}{4} + \frac{1}{16} + \frac{2}{16} = 0.9375$.

The Proposition 1 shows that the lower and upper bounds are tighter over the iterations: the more iterations we run, the closer we are to ω.

Proposition 1
For any $t \geq 1$, we have:

$$\underline{\Omega_{t-1}} \leq \underline{\Omega_t} \tag{11}$$

$$\overline{\Omega_t} \leq \overline{\Omega_{t-1}} \tag{12}$$

Proof
For Equation (11), from $\underline{\Omega_0} = 0$ and $\frac{\left|P_t^{\neg\text{mon}}\right|}{|P_t|} \geq 0$ and $\underline{\Omega_t} = \underline{\Omega_{t-1}} + \frac{\left|P_t^{\neg\text{mon}}\right|}{|P_t|}$, we can deduce $\underline{\Omega_{t-1}} \leq \underline{\Omega_t}$.

Then, to prove Equation (12), we need first to state an invariant of Algorithm 1, Line 7:

$$\left(P_t^{\text{partially mon}} \cup P_t^{\neg\text{mon}}\right) \subseteq \widehat{P}_t \tag{13}$$

Based on this, since $\widehat{P}_t$ is a finer partition of $P_{t-1}^{\text{partially mon}}$, the following inequality is true by construction:

$$\left|P_t^{\text{partially mon}} \cup P_t^{\neg\text{mon}}\right| \leq \left|\widehat{P}_t\right| \leq \left|P_{t-1}^{\text{partially mon}}\right| * \frac{|P_t|}{|P_{t-1}|}$$

We divide both sides of the inequality by $|P_t|$ and add $\underline{\Omega_{t-1}}$:

$$\underline{\Omega_{t-1}} + \frac{\left|P_t^{\text{partially mon}} \cup P_t^{\neg\text{mon}}\right|}{|P_t|} \leq \underline{\Omega_{t-1}} + \frac{\left|P_{t-1}^{\text{partially mon}}\right|}{|P_{t-1}|}$$

Since $P_t^{\text{partially mon}} \cap P_t^{\neg\text{mon}} = \emptyset$, we have:

$$\underline{\Omega_{t-1}} + \frac{\left|P_t^{\neg\text{mon}}\right|}{|P_t|} + \frac{\left|P_t^{\text{partially mon}}\right|}{|P_t|} \leq \overline{\Omega_{t-1}}$$

therefore by Equation (9):

$$\underline{\Omega_t} + \frac{\left|P_t^{\text{partially mon}}\right|}{|P_t|} \leq \overline{\Omega_{t-1}}$$

and therefore by Equation (10):

$$\overline{\Omega_t} \leq \overline{\Omega_{t-1}}$$ □

3.3.4 Monotony error estimation

If the output of the network is a real, it is useful to produce an estimation of the seriousness of the monotony violation on the sub-spaces where monotony does not hold. If we refer to Fig. 5, the error corresponds to the height of the volume represented in green. Its measure unit is the same as that of the network output, which generally has a physical interpretation. For the example considered in Sect. 4, it is a distance measured in meters.

The interesting measure here is the maximum bound on the error for each subspace from $P^{\neg\text{mon}} \cup P^{\text{partially mon}}$. The method presented in the previous section can compute this

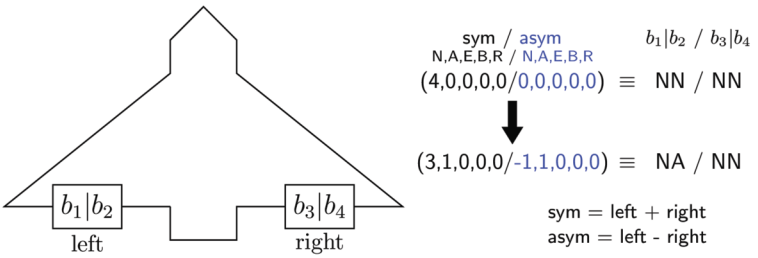

Fig. 8 Representation of the position of the four brakes on an aircraft, denoted by $b_i \in \{N, A, E, B, R\}$. For example, we have NN-NN when all the brakes are in the normal state. Then if the state of one of the left brakes becomes *Altered*, we have NA-NN. Note that NA-NN≡AN-NN due to the choice of the representation of the brakes

bound on its value. Indeed, the call to the MILP solver on line 4 of Algorithm 2 can be set to use $f(x_1) - f(x_2)$ as an optimization objective. Then, when the answer is SAT, the call also produces an approximate value and a hard upper bound for $\mathbf{max}(f(x_1) - f(x_2))$.

4 Case study: braking distance estimation

4.1 Description of the case study

Our case study is an R&D project from the aeronautical industry consisting in training a neural network to estimate the braking distance of an aircraft based on physical information. The R&D team has provided us with a trained feedforward neural network composed of 2 layers (30 and 29 neurons in the first and second layer, respectively) and ReLU activation functions. There are 15 input features, including 13 discrete and 2 continuous. For example, one of the continuous features models the aircraft gross weight (normalized to the [0,1] interval). The discrete features capture information such as the state of the runway (dry/wet), the state of the spoilers or the state of the brakes. Some features are binary, while others take an integer value in a set capturing the various possible configurations. For example, the state of the 4 brakes of the aircraft is described through 10 input features, as follows. Each brake has 5 possible modes: Normal (N), Altered (A), Emergency (E), Burst (B), and Released (R). The network has 2 features for each brake mode: *(i)* the total number of brakes in a given mode (referred to as "symmetric") and *(ii)* the difference between the number of brakes on the left and right side of a given mode (referred to as "asymmetric") (see Fig. 8). Thus, there is an equivalence between an allowed value for the 10 input features and the state of the pairs of brakes on the left and right sides of the aircraft, although the state of each individual brake cannot be retrieved. For clarity, we will describe the state of the brakes using the form "b_1b_2-b_3b_4", where b_1b_2 and b_3b_4 should be seen as unordered pairs, since the order is not captured by the input features. There are in total 225 distinct brake states.

Note that this particular encoding of the brakes' states implies some specific constraints on the 10 input features. For each possible brake mode (N, A, E, B, R), the "symmetric" feature is comprised between 0 and 4 and the "asymmetric" feature between −2 and 2. Moreover, the sum of the 5 "symmetric" features is necessarily 4, and the sum of the "asymmetric" features is 0. In addition there are parity constraints: if for a certain mode the "symmetric" feature is odd, the "asymmetric" feature is either −1 or 1, and if the "symmetric" feature is even, the "asymmetric" feature is also even. These constraints needed to be added to the MILP model in order to ensure that the SAT answers provided by the solver during the run of our verification algorithm are not spurious. Due to their very specific nature, they had to be hand-coded; it should also be noted that the parity constraints are not linear, but could be modeled using so-called *indicator* constraints.[2]

Several monotony properties are important for this network, some of which are listed below:[3]

1. The braking distance on a wet runway is larger than on a dry runway;
2. The braking distance increases depending on the state of the spoilers (there is an order for the different configurations of the spoilers);
3. Over a certain threshold, the braking distance increases with the aircraft weight;
4. The braking distance increases when the brakes' state deteriorates.

Properties 1–3 listed above involve only one input feature, which can be discrete (properties 1 and 2) or continuous (property 3). In the following, we concentrate on property 4, which involves all 10 input features that model the brakes' states simultaneously. To perform the monotony analysis, we need to define what *deteriorates* means formally. Relying on the system expert's knowledge, the following partial order

[2] A kind of constraint supported by many MILP solvers, which takes the form $boolean\_var \implies affine\_inequality$.

[3] All the properties should be read as if prefixed with "all else being equal".

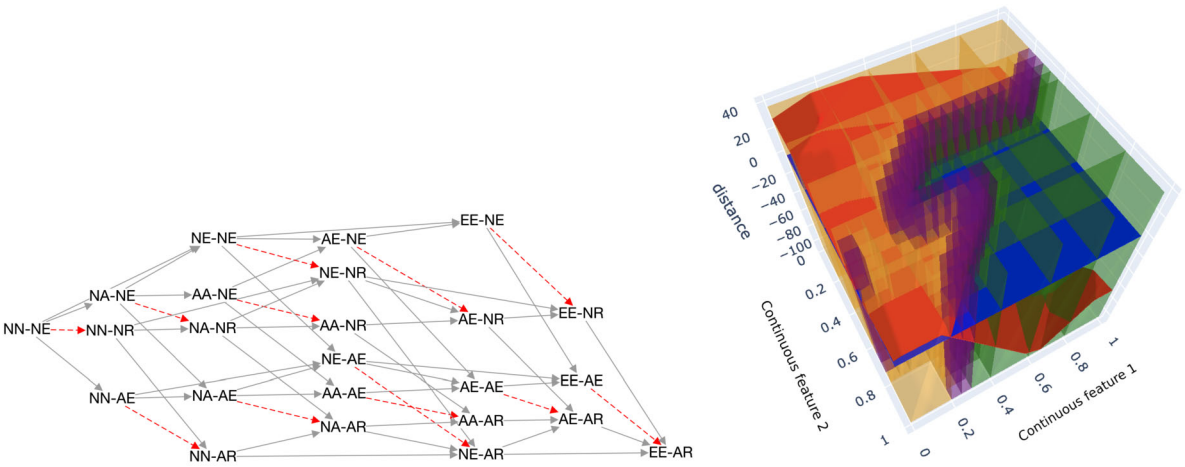

Fig. 9 Example of visualization of features in α (left) and $\bar{\alpha}$ (right) (Color figure online)

applies to the different modes of the brake:

$$N \prec_b A \prec_b E \prec_b B \prec_b R, \tag{14}$$

where $b_i \prec_b b_j$ means that the state b_j is more deteriorated than the state b_i.

We can easily extend the partial order $\preceq_b$ on one brake to the state of an aircraft's brakes composed of 4 brakes. Let $S_1 = (b_1, b_2, b_3, b_4)$ and $S_2 = (b'_1, b'_2, b'_3, b'_4)$ be two states of an aircraft's brakes, we have

$$\begin{aligned} S_1 \prec S_2 \quad \Longleftrightarrow \quad & \forall b_i, b'_i \in S_1 \times S_2,\ b_i \preceq_b b'_i \text{ and} \\ & \exists b_i, b'_i \in S_1 \times S_2\ \ b_i \prec_b b'_i \end{aligned} \tag{15}$$

It means that S_2 is deteriorated compared to S_1 if and only if, for all brakes in S_2, the brake's mode in S_2 is at most as good as its counterpart in S_1 and there exists a brake in S_2 whose mode is strictly worse than its counterpart in S_1.

4.2 Experimentation

4.2.1 Setup

Let $\mathcal{V}$ be the set of 15 input features described above, $X = \times_{v \in \mathcal{V}} \mathcal{D}(v)$ be the input space, $Y = \mathbb{R}^+$ be the output space, and $f : X \mapsto Y$ be the neural network estimating the braking distance of an aircraft. We consider the monotony property as formulated in Definition 1. As stated earlier, we are dealing with the monotony with respect to the brakes' space. Hence, $\alpha \subseteq \mathcal{V}$ is made up of the ten features describing the state of the brakes and the partial order $\preceq$ on $\times_{v \in \alpha} \mathcal{D}(v)$ is as defined in Equation (15). We take advantage of the discrete nature of the brakes' features: a natural partition $\mathcal{P}$ is to enumerate all the possible values for the ten brakes features. We have $|\mathcal{P}| = 225$.

4.2.2 Monotony analysis

We run Algorithm 1 for 5 iterations with the setup described above. The algorithm is explained in Sect. 3.3. Here we only focus on the partitions used for the analysis and the refinement strategy specific to the case study. Additionally, we will see how to capitalize on the data available at each iteration to perform some space exploration. P_1 is setup using $\mathcal{P}$, and $\preceq$; it represents the brakes' sub-space. Then our partition's refining strategy is to start with the remaining discrete features (second iteration) and then consider the continuous features (the last three iterations). For the discrete features, the partitioning consists in enumerating the possible values, while for the continuous features, it consists of a uniform partition (finer through the iterations). To illustrate the impact of the refinement on the level of details of the analysis, we detail the total number of sub-spaces in each partition P_t: $|P_1| = 10{,}800$, $|P_2| = 259{,}200$, $|P_3| = 6{,}480{,}000$, $|P_4| = 25{,}920{,}000$ and $|P_5| = 103{,}680{,}000$.

Based on the partition and the outcomes of $\mathbf{F}(\cdot)$, the algorithm yields at each iteration the metrics $\underline{\Omega_t}$ and $\overline{\Omega_t}$. Nonetheless, for our case study, we put in place visualization means (see Fig. 9). However, the relevant visualizations helping space exploration are case-dependent, so we do not propose any generic way to do it. Firstly, it might be relevant to visualize the sub-space composed of the features on which the partial order $\preceq$ is defined, i.e., α corresponding to the brakes' space. It is modeled as a graph where the nodes are the brakes' states (the elements of $\mathcal{P}$), and the edges are the transitions between the states modeled by the partial order $\prec$ (the elements of P_1), and with the outcomes of the first iteration, we can highlight (dashed line in Fig. 9) the transitions that violate the monotony property (in Fig. 9, we plot only a sub-graph as an example). Then, to include the information of the formal verification of the following iteration

Table 2 Values of $\underline{\Omega}$ and $\overline{\Omega}$ bounding the percentage of the space where the monotony property is violated

Metrics	It.1	It.2	It.3	It.4	It.5
$\overline{\Omega_t}$	11.57%	4.11%	1.95%	1.72%	1.61%
$\underline{\Omega_t}$	0.03%	0.45%	1.12%	1.29%	1.39%

in the space visualization, we plot some features versus the difference of distances $f(x_1) - f(x_2)$ and visualize in which sub-spaces monotony issue occurs. These visualizations are helpful for exploration purposes after the analysis for the expert of the system (e.g., if the expert can identify some place of interest within the space and wants to know what happens there).

4.2.3 Metrics: non-monotonic space coverage

At each iteration, we compute $\underline{\Omega_t}$ and $\overline{\Omega_t}$, which bound the ratio of the space where f violates the monotony property (i.e., ω). The results are summarized in Table 2. In Fig. 10, we can clearly see the convergence of $\underline{\Omega_t}$ and $\overline{\Omega_t}$. At the first iteration, we can explain the notable gap between $\underline{\Omega_1}$ and $\overline{\Omega_1}$ by the coarse partitioning of the space. That is why, $\overline{\Omega_1}$ is large (numerous sub-spaces where the monotony property is partially respected) and $\underline{\Omega_1}$ small (only few sub-spaces where the monotony property does not hold). We can notice a significant drop of $\overline{\Omega_t}$ compared to the rise of $\underline{\Omega_t}$: there are more sub-spaces where the monotony property holds than not. Eventually the algorithm yields a narrow gap between the bounds; we obtain at the fifth iteration: $1.39\% \leq \omega \leq 1.61\%$. The stopping criterion of the algorithm may depend on various things such that the system's requirements (e.g. bounds precision, max value to not cross for $\underline{\Omega}$ or min value to reach for $\overline{\Omega}$).

Through these five steps, we analyze the monotony of f considering finer and finer partition of the space; we obtain: *(i)* metrics bounding the percentage of the space where the neural network is non-monotonic and *(ii)* the identification of the sub-spaces where the monotony issue occurs thanks to the formal verification on each elements of the partitions.

We run our experiments on MacBook Pro 8 core 2,3 GHz Intel Core i9 with 32 Gb of RAM. The MILP solver used is Gurobi 9 [15] and our monotony analysis for the property discussed in this section took less than 10 hours.

4.3 Discussion

The analysis and visualization techniques described above allowed to detect that, in the studied example, the monotony violations are concentrated close to the boundary of the operational design domain for one of the features (the aircraft weight). The possible mitigation actions are to re-train the model using a more representative dataset for these areas of the ODD, or to restrict the use of the model for a subareas of the ODD for which it is deemed safe. The followup actions are ongoing and are the responsibility of the team that provided the model.

Although the monotony analysis is not sufficient for ensuring the correctness of the outputs of the neural network, it is a necessary complement to other neuronal network validation techniques. We expect monotony analysis results to be included in the assurance cases of ML items in future certification procedures, along with other techniques (some yet to be invented) for addressing the challenges identified in Sect. 2.3, in particular robustness. Existing EUROCAE/RTCA standards for software items (such as DO-178C) define requirements on safety levels and bounds on error occurrence frequency depending on the software item criticality. In a similar manner, we anticipate the certification will adjust what is considered an acceptable output of a neural network to an error bound depending on the criticality of its context.

5 Related work

In recent years, assessing the robustness of neural networks has been tackled with formal verification (i.e., sound algorithms demonstrating whether a property holds or not). Verifying properties on a neural network is challenging because neural networks are non-convex functions with many non-linearities and hundreds to millions of parameters. Even if the type of architecture studied is restricted to piece-wise linear networks, it is known that this problem is already NP-hard [40]. There has been tremendous progress in the field of verification, from robustness proofs of networks trained on MNIST to scaling up to CIFAR 10 and even TinyImagenet. These successes are particularly due to the collaboration of different communities that made it possible to formulate and tackle the robustness problem from different perspectives. Without being exhaustive, we can cite the methods that rely on Lipschitz-based optimization [43, 44], input refinement [38] and semi-definite relaxations [29].

So far, the verification community has mainly tackled robustness verification from adversarial robustness [36] to computing the reachable set of a network [41] despite a few other properties that are highly relevant for the industry. Among those properties, partial monotony under specific inputs appears to be a key property, especially for regression tasks. Indeed, the need for monotony appeared in various contexts such as surrogate modeling [16], economics [11], fairness [18], or interpretability [26] and is thus a highly desirable feature in the industry. Previous works proposed to enforce the monotony property during the design. Gupta et al

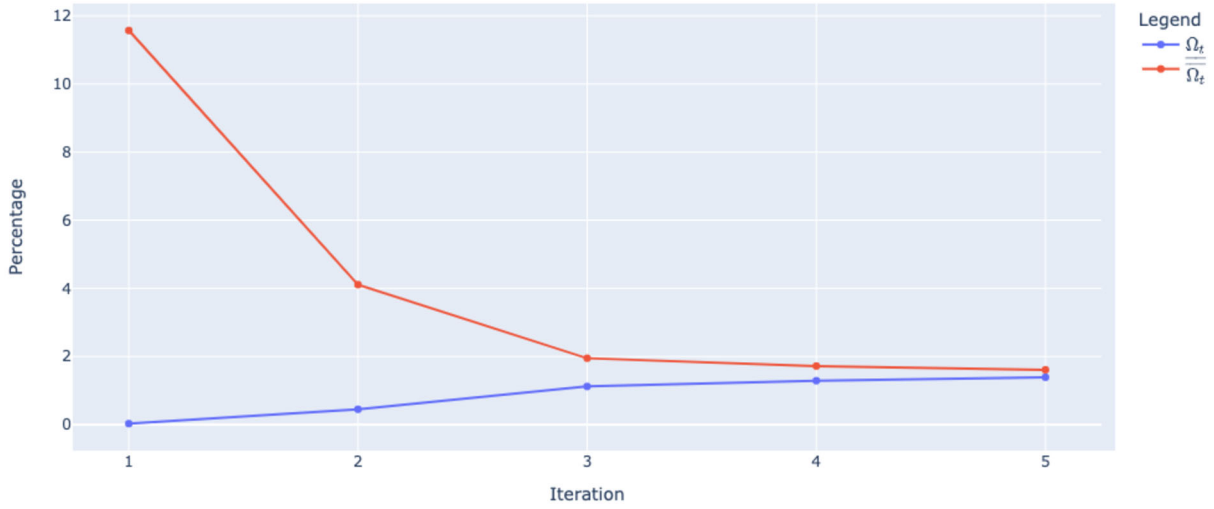

Fig. 10 Evolution of $\underline{\Omega_t}$ and $\overline{\Omega_t}$.

[14] and Gauffriau et al [12] rely on heuristics monotonic regularizers or hand-designed monotonic architectures, whose main drawback lies in the absence of guarantees and therefore will require verification as a post-processing step, e.g., using a method such as the one proposed in this paper. On the other side, Liu et al [20] adopts an approach also based on MILP encoding to verify monotonicity, but concentrate on using it for a different aim than analyzing (non-)monotonic space coverage. The goal in their case is hardening training, which can prove to be costly in practice. They also provide an idea for improving scalability of monotonicity verification for deep neural networks by decomposing the network into a stack of two-layer networks and then verifying their monotonicity separately. Indeed, this provides a sufficient (but not necessary) condition for global monotonicity, and the idea could be used in conjunction to our method, but that is beyond the scope of the present paper.

An important difference between our approach and the approach from Liu et al [20] is that their approach applies only to monotony constraints on continuous inputs, whereas ours also applies to discrete inputs, such as the ones appearing in our case study. For continuous inputs, monotony is equivalent to verifying a property on the gradients on the whole domain. Indeed the sign of the gradient component corresponding to monotonous inputs should always keep the same sign, positive or negative. However, for a neural network with discrete inputs, the gradient sign condition is *not necessary* for the monotony to hold, even when the gradient can be computed by extending the input domain to reals. For piece-wise linear neural networks such as ReLU networks, we can base verification on the very definition of monotony (Definition 1), which can be cast as solving a mixed-integer linear programming problem. This method is complementary to the literature using the gradient condition and can verify monotony over discrete inputs.

To our knowledge, most previous work has considered monotony under continuous inputs, while many industrial use-cases have monotony constraints on discrete inputs. One notable exception is the fairness verification in the work of Urban et al [37] that can be applied on both a discrete or a continuous input and has similarities with monotony verification.

Verifying the monotony is recognized to be more challenging than robustness, since it is a global property on the whole domain rather than a local neighborhood [20]. However, we argue that applying partial monotony over the whole domain, which may affect the performance and put at risk the product's release, is a very drastic approach. Indeed, in an industrial context, it is necessary to balance quality and safety, especially as the systems are not only constrained by monotony, but also by other specifications such as accuracy. The solution we propose is a partitioning scheme that splits the operational domain into areas where the monotony property is respected and areas where it is (partially) violated; in the latter, the neural network's behavior could be mitigated. This possibility has been considered on a collision detection use case [7] and studied at a higher level for the certification of a system before an ML component [22].

6 Conclusion

In this paper, we discuss the challenges raised by including components based on machine learning in critical systems. We focus on the impact it has on well established certification processes, in particular for civil aviation. Some of the

challenges listed can be overcome with the help of formal verification techniques.

We then focus on the problem of formally verifying monotony properties on feed-forward neural networks with ReLU units. We propose an iterative method based on MILP that allows to identify the input sub-spaces where the neural network does not satisfy the property. The verification procedure is suited for both discrete and continuous features. Our proposal is a step further in the demonstration that neural networks can preserve important functional properties and therefore in the capability to embed the ML technology in an aeronautical safety-critical system.

We applied this method on an aeronautical case study that consists in estimating the braking distance of an aircraft using a neural network mixing discrete and continuous inputs. We managed to quantify the percentage of the space where the neural network does not preserve the monotony property and to identify formally each sub-space where it occurs. In addition, we showed that we can capitalize on the available data to visualize the sub-spaces for helping the braking function's experts in processing the results of the algorithm.

Note that this work leaves room for some optimizations, such as using tighter big-M values in Equations (2)-(3), or using asymmetric bounds, computed by incomplete methods such as [34, 42]. Initial attempts to integrate such bounds in our method proved to be less effective than the big-M method in terms of MILP solving time, but this needs to be further explored. To the best of our knowledge, the scalability of complete methods remains a challenge in the verification community and is mainly used with "shallow" neural networks. Thus, this method is mainly useful for small to medium networks used as surrogates or for control.

As an extension of this work, we plan to estimate the integral under the curve of $f(x_1) - f(x_2)$ in the sub-spaces where the monotony is violated by leveraging our definition of the monotony property. This would give a key indicator on the level of violation of the monotony property that could support the performance of the training phase. Another area for future work concerns experimenting with different partition functions, as the proposed method is independent of the choice of the partition function.

References

1. Amershi, S., Begel, A., Bird, C., et al.: Software engineering for machine learning: a case study. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), pp. 291–300 (2019). https://doi.org/10.1109/ICSE-SEIP.2019.00042
2. Biannic, J., Hardier, G., Roos, C., et al.: Surrogate models for aircraft flight control: some off-line and embedded applications. Aerosp. Lab. **12**, 1 (2016)
3. Carlini, N., Wagner, D.A.: Towards evaluating the robustness of neural networks. In: IEEE SP. IEEE Computer Society, pp. 39–57 (2017). https://doi.org/10.1109/SP.2017.49
4. Chen, S., Sun, Y., Li, D., et al.: Runtime safety assurance for learning-enabled control of autonomous driving vehicles. In: 2022 International Conference on Robotics and Automation (ICRA), pp. 8978–8984 (2022). https://doi.org/10.1109/ICRA46639.2022.9812177
5. Cheng, C.H., Nührenberg, G., Ruess, H.: Maximum resilience of artificial neural networks. In: D'Souza, D., Narayan Kumar, K. (eds.) Automated Technology for Verification and Analysis, pp. 251–268. Springer, Berlin (2017). https://doi.org/10.1007/978-3-319-68167-2_18
6. Cofer, D.D., Amundson, I., Sattigeri, R., et al.: Run-time assurance for learning-enabled systems. In: Lee, R., Jha, S., Mavridou, A. (eds.) NASA Formal Methods – 12th International Symposium, NFM 2020, Moffett Field, CA, USA, May 11–15, 2020. Proceedings, Lecture Notes in Computer Science, vol. 12229, pp. 361–368. Springer, Berlin (2020). https://doi.org/10.1007/978-3-030-55754-6_21
7. Damour, M., Grancey, F.D., Gabreau, C., et al.: Towards certification of a reduced footprint ACAS-Xu system: a hybrid L-based solution. In: Proceedings, Computer Safety, Reliability, and Security – 40th International Conference,SAFECOMP 2021, York, UK, September 8-10, 2021, pp. 34–48 (2021). https://doi.org/10.1007/978-3-030-83903-1_3
8. de Moura, L.M., Bjørner, N.S.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008. Budapest, Hungary, March 29-April 6, 2008. Proceedings, Lecture Notes in Computer Science, vol. 4963, pp. 337–340. Springer, Berlin (2008). https://doi.org/10.1007/978-3-540-78800-3_24
9. EASA: CS-25 Amendment 27 (2021). https://www.easa.Europa.eu/downloads/136622/en
10. EASA: EASA Concept Paper: First usable guidance for Level 1 machine learning applications (2021). https://www.easa.Europa.eu/downloads/134357/en
11. Feelders, A.J.: Prior knowledge in economic applications of data mining. In: European Conference on Principles of Data Mining and Knowledge Discovery, pp. 395–400. Springer, Berlin (2000). https://doi.org/10.1007/3-540-45372-5_42
12. Gauffriau, A., Malgouyres, F., Ducoffe, M.: Overestimation learning with guarantees (2021). arXiv preprint arXiv:2101.11717
13. Grossmann, I.E.: Review of nonlinear mixed-integer and disjunctive programming techniques. Optim. Eng. (2002)
14. Gupta, A., Shukla, N., Marla, L., et al.: How to incorporate monotonicity in deep networks while preserving flexibility? (2019). arXiv preprint arXiv:1909.10662
15. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2022). https://www.gurobi.com
16. Hao, J., Ye, W., Jia, L., et al.: Building surrogate models for engineering problems by integrating limited simulation data and monotonic engineering knowledge. Adv. Eng. Inform. **49**, 101342 (2021). https://doi.org/10.1016/j.aei.2021.101342
17. Jian, Z.D., Chang, H.J., Ts, H., et al.: Learning from simulated world – surrogates construction with deep neural network. In: SIMULTECH 2017: Proceedings of the 7th International Conference on Simulation and Modeling Methodologies, Technologies and Applications. SCITEPRESS (2017). https://doi.org/10.5220/0006418100830092
18. Karpf, J.: Inductive modelling in law: example based expert systems in administrative law. In: Proceedings of the 3rd International Conference on Artificial Intelligence and Law, pp. 297–306 (1991). https://doi.org/10.1145/112646.112684
19. Katz, G., Huang, D.A., Ibeling, D., et al.: The marabou framework for verification and analysis of deep neural networks. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification, pp. 443–452.

Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_26
20. Liu, X., Han, X., Zhang, N., et al.: Certified monotonic neural networks. Adv. Neural Inf. Process. Syst. **33**, 15427–15438 (2020). https://proceedings.neurips.cc/paper/2020/hash/b139aeda1c2914e3b579aafd3ceeb1bd-Abstract.html
21. Madry, A., Makelov, A., Schmidt, L., et al.: Towards deep learning models resistant to adversarial attacks. In: ICLR. OpenReview.net (2018). https://openreview.net/forum?id=rJzIBfZAb
22. Mamalet, F., Jenn, E., Flandin, G., et al.: White Paper Machine Learning in Certified Systems (2021). https://hal.archives-ouvertes.fr/hal-03176080
23. Marques-Silva, J., Ignatiev, A.: Delivering trustworthy AI through formal XAI. In: Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 – March 1, 2022, pp. 12342–12350. AAAI Press, Menlo Park (2022). https://ojs.aaai.org/index.php/AAAI/article/view/21499
24. Martin, R.: Assured software – a journey and discussion (2017). https://www.his-2019.co.uk/session/cwe-cve-its-history-and-future
25. Müller, M.N., Makarchuk, G., Singh, G., et al.: PRIMA: general and precise neural network certification via scalable convex hull approximations. Proc. ACM Program. Lang. **6**(POPL), 43 (2022). https://doi.org/10.1145/3498704
26. Nguyen, A., Martínez, M.R.: MonoNet: towards interpretable models by learning monotonic features (2019). arXiv preprint arXiv:1909.13611
27. Phillips, P.J., Hahn, C., Fontana, P., et al.: Four principles of explainable artificial intelligence (2021). https://doi.org/10.6028/NIST.IR.8312. https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=933399
28. Picard, S., Chapdelaine, C., Cappi, C., et al.: Ensuring dataset quality for machine learning certification. In: ISSRE, pp. 275–282 (2020). https://doi.org/10.1109/ISSREW51248.2020.00085
29. Raghunathan, A., Steinhardt, J., Liang, P.S.: Semidefinite relaxations for certifying robustness to adversarial examples. In: Advances in Neural Information Processing Systems, pp. 10877–10887 (2018). https://proceedings.neurips.cc/paper/2018/hash/29c0605a3bab4229e46723f89cf59d83-Abstract.html
30. Rushby, J.: The interpretation and evaluation of assurance cases. Tech. Rep., (2015) http://www.csl.sri.com/users/rushby/papers/sri-csl-15-1-assurance-cases.pdf
31. Schweiger, A., Annighoefer, B., Reich, M., et al.: Classification for avionics capabilities enabled by artificial intelligence. In: 2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC), pp. 1–10 (2021). https://doi.org/10.1109/DASC52595.2021.9594364
32. Singh, G., Gehr, T., Püschel, M., et al.: Robustness certification with refinement. In: International Conference on Learning Representations (2019). https://openreview.net/forum?id=HJgeEh09KQ
33. Sudakov, O., Koroteev, D., Belozerov, B., et al.: Artificial neural network surrogate modeling of oil reservoir: a case study. In: International Symposium on Neural Networks, pp. 232–241. Springer, Berlin (2019). https://doi.org/10.1007/978-3-030-22808-8_24
34. Tjeng, V., Xiao, K.Y., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: ICLR (2019). https://openreview.net/forum?id=HyGIdiRqtm
35. Tsuzuku, Y., Sato, I., Sugiyama, M.: Lipschitz-margin training: scalable certification of perturbation invariance for deep neural networks. In: NeurIPS, pp. 6542–6551 (2018). https://proceedings.neurips.cc/paper/2018/hash/485843481a7edacbfce101ecb1e4d2a8-Abstract.html
36. Urban, C., Miné, A.: A review of formal methods applied to machine learning (2021). arXiv preprint arXiv:2104.02466. https://arxiv.org/abs/2104.02466
37. Urban, C., Christakis, M., Wüstholz, V., et al.: Perfectly parallel fairness certification of neural networks. Proc. ACM Program. Lang. **4**(OOPSLA), 185 (2020). https://doi.org/10.1145/3428253
38. Wang, S., Pei, K., Whitehouse, J., et al.: Formal security analysis of neural networks using symbolic intervals. In: 27th USENIX Security Symposium (USENIX Security, vol. 18, pp. 1599–1614. USENIX Association, Baltimore (2018). https://www.usenix.org/conference/usenixsecurity18/presentation/wang-shiqi
39. Wang, S., Zhang, H., Xu, K., et al.: Beta-CROWN: efficient bound propagation with per-neuron split constraints for neural network robustness verification. In: Advances in Neural Information Processing Systems (2021). https://proceedings.neurips.cc/paper/2021/hash/fac7fead96dafceaf80c1daffeae82a4-Abstract.html
40. Weng, T., Zhang, H., Chen, H., et al.: Towards fast computation of certified robustness for relu networks. In: ICML. Proceedings of Machine Learning Research (2018). http://proceedings.mlr.press/v80/weng18a.html
41. Xiang, W., Tran, H.D., Johnson, T.T.: Output reachable set estimation and verification for multilayer neural networks. IEEE Trans. Neural Netw. Learn. Syst. **29**(11), 5777–5783 (2018). https://doi.org/10.1109/TNNLS.2018.2808470
42. Xu, K., Shi, Z., Zhang, H., et al.: Automatic perturbation analysis for scalable certified robustness and beyond. In: NeurIPS, pp. 1129–1141 (2020). https://proceedings.neurips.cc/paper/2020/hash/0cbc5671ae26f67871cb914d81ef8fc1-Abstract.html
43. Zhang, H., Weng, T.W., Chen, P.Y., et al.: Efficient neural network robustness certification with general activation functions. In: Advances in Neural Information Processing Systems, pp. 4939–4948 (2018). https://proceedings.neurips.cc/paper/2018/hash/d04863f100d59b3eb688a11f95b0ae60-Abstract.html
44. Zhang, H., Zhang, P., Hsieh, C.J.: Recurjac: an efficient recursive algorithm for bounding Jacobian matrix of neural networks and its applications. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 5757–5764 (2019). https://doi.org/10.1609/aaai.v33i01.33015757