How to Verify Generalization Capability of a Neural Network with Formal Methods

Arthur Clavière, Dmitrii Kirov, and Darren Cofer

Collins Aerospace
`name.lastname@collins.com`

Abstract Generalization of a machine learning (ML) model is its capability to maintain desired performance on input data to which it was not exposed during training. A bound on the model generalization error can provide important evidence of the absence of unintended behavior of the model, which is the key requirement for safety-critical systems and software. Such bounds are typically estimated statistically and provide a level of confidence that the bound holds. In this paper, we show how ML model generalization capability and bound can be assessed using *formal* methods providing a rigorous mathematical guarantee. We focus on applications that use neural networks to approximate a function with a low-dimensional, well-defined and bounded input space. We propose an iterative procedure that starts with partitioning the neural network input space into regions using one or multiple resolutions. Within each region, we formalize a property that the error made by the neural network on *any* data point inside the region is below a given tolerance. Proving such property provides a formal generalization guarantee for a given region. We employ an abstract interpretation solver to verify these properties over the entire input space partition. We iteratively refine the regions in which the proof could not be achieved by sampling or generating new data, forming a new local partition with higher granularity. This refinement follows a heuristic that aims to minimize the amount of new data to be produced. We demonstrate our methodology by proving generalization capability of a neural network-based avionics function.

Keywords: Neural Networks · Verification · Formal Methods.

1 Introduction

Machine Learning (ML) is an enabling technology in aviation. ML models, such as neural networks (NNs), can be used to develop advanced avionics algorithms for decision making, planning, and perception. They can also efficiently approximate existing software, reducing its memory and computational costs [13]. Such ML-enabled components and systems are typically safety-critical. Therefore, they must undergo rigorous design assurance and certification processes to be deployed onboard aircraft. Data-driven ML approaches are not fully amenable to existing standards for assuring safety-critical software, such as DO-178C [19], due to several gaps [20]. Aviation industry together with global certification authorities has made significant progress towards new ML-specific standard and guidance that is expected to appear in the next few years [21]. This standard will cover assurance objectives that are specific to ML.

One of the key ML assurance objectives is to provide evidence of the *generalization* capability of the ML model, i.e., its capability to maintain desired performance on input data to which it was not exposed during training. Such evidence must be provided over the entire Operational Design Domain (ODD), i.e., space of inputs where the model is designed to operate. The concept of generalization is not new in the ML community, and the widely accepted best practice for assessing it is to measure the *in-sample* error of the ML model (also called empirical risk) using a test dataset. While in-sample error can help to detect overfitting problems, it does not provide knowledge on how the model would perform on *unseen* data that may occur during operation. Such *out-of-sample* inputs may be sources of unintended behavior of the model, which is not acceptable in safety-critical aviation applications. The European Union Aviation Safety Agency (EASA) in their AI Concept Paper [3] suggests statistical learning theory methods to estimate model generalization bounds. Similar objectives appear in the upcoming ML assurance standard ARP6983 [21] jointly developed by SAE G-34 and EUROCAE WG-114 working groups. The Federal Aviation Administration (FAA) also requests to measure both the empirical error and the level of confidence that this error holds over the entire input domain [18].

While statistical methods can instrument generalization assessment, it is not yet clear whether a probabilistic measure of the bound would be acceptable, especially in high-criticality ML applications. In this paper, we consider a different perspective by looking for more rigorous, formal generalization guarantees for ML models. To provide a useful and tangible result, we focus on low-complexity NN models (tens to thousands learnable parameters) with low-dimensional (in the order of 10) well-defined and bounded input spaces. Such a class of NN applications is very well represented in the aviation domain (ACAS-XU is one example). We employ formal methods, which recently have been shown to be effective in ML assurance [10], to define and verify the NN generalization property.

Our approach is an iterative procedure that starts with partitioning the NN input space into regions using one or multiple resolutions. We call these regions “hyperrectangles”. Their corners correspond to labeled data points from the existing dataset, which allows us to formalize a local property that the error made by the NN on *any* data point inside the hyperrectangle is below a given tolerance. Proving such property provides a formal generalization guarantee for a hyperrectangle, essentially representing a bound on the out-of-sample error. We employ an abstract interpretation solver to verify these properties over the entire input space partition. We iteratively refine hyperrectangles in which the proof could not be achieved by sampling or generating new data, forming a new local partition with higher granularity. This refinement follows a heuristic that aims to minimize the amount of new data to be produced.

Our contributions can be summarized as follows:

- We propose a mathematical formalization of the NN generalization property, and a method to prove this property over the entire input domain using NN verifiers with abstract interpretation. We discuss the assumptions and the limitations of the approach.
- We develop a verification procedure that iteratively refines regions where the property could not be proven valid (e.g., due to over-approximations used in the verifiers) to maximize the volume of the input space where generalization proof could be obtained.
- We demonstrate our approach by assessing generalization of a NN-based advisory system for pilots, which is a safety-critical avionics function.

Our method is currently applicable to NNs with low-dimensional input spaces (indicatively, number of inputs in the order of 10). Such NNs have a lot of applications in safety-critical aviation, as well as other domains. We assess the computational complexity of our algorithm in the paper, and discuss several performance improvements to enhance scalability.

2 Background

2.1 Neural Network Generalization Assessment

The purpose of generalization assessment is to see how well the NN performs on the data that it was not exposed to during training. Traditional practice suggests using a test (holdout) dataset to measure the generalization capability of the model. Test dataset allows to measure a so-called *empirical risk*, i.e., to understand the performance of the NN using the examples (test points) available in the dataset. Emerging aviation guidance, such as EASA AI Concept Paper [3], goes beyond this and prescribes an analytical bound on the generalization error to be estimated. The goal is to estimate a worst-case error that the NN (more generally, the ML model) could make on *any* unseen data within its ODD. It is called the *out-of-sample error* (or theoretical risk), while the empirical risk on the test dataset is called the *in-sample error*. The difference between the two is the *generalization gap* [9]. Statistical bounds are typically used to assess generalization of ML models, while this paper proposes an alternative approach using formal methods.

2.2 Neural Network Formal Verification

Formal analysis of NNs is focused mainly on robustness verification i.e., checking that a perturbation of an input yields a bounded deviation of the outputs [10]. To verify such properties, several formal verification approaches and solvers were introduced. These approaches can be roughly split into complete methods (e.g., see [23,11,25]) and incomplete methods (e.g., see [26,12,22,24]). The former provide sound and complete verification, i.e., can always prove or disprove the property, but incur higher computational complexity. The latter rely on sound over-approximations and trade completeness (i.e., can return *Unknown* answers) with higher performance. Improvements in state-of-the-art NN verifiers can be observed from the results of the annual Verification of NNs Competition (VNNComp) [5], where we also submitted several benchmark problems over the recent years [14,15].

3 Verification Method

Our proposed method applies to *regression problems* i.e., when the NN outputs values in $\mathbb{R}$. In the remainder of this section, for the sake of simplicity, we consider a NN with a single output (Remark 1 explains how the method can be generalized to multiple outputs).

We adopt the following notation:

- $I = \times_{i=1,\dots,N}[l_i, u_i]$ is the Operational Design Domain (ODD) of the NN, captured as a Cartesian product of admissible ranges along the input dimensions, where l_i and u_i are, respectively, the lower and the upper bound of the i-th dimension (feature), and N is the dimensionality of I;
- $\mathcal{N} : I \to \mathbb{R}$ is the NN to be verified;
- $f : I \to \mathbb{R}$ is the ground truth function, i.e., the target function to be approximated with $\mathcal{N}$;
- $\mathbf{H} = (H_1, \dots, H_Q)$ is a set of hyperrectangles that form a partition of I;
- $H_j = \times_{i=1,\dots,N}[l_{i,j}, u_{i,j}]$ is the j^{th} hyperrectangle in $\mathbf{H}$;
- $\mathcal{N}(H_j) = [l_{\mathcal{N},j}, u_{\mathcal{N},j}]$ is the range of outputs of $\mathcal{N}$ corresponding to input points contained in H_j;
- $f(H_j) = [l_{f,j}, u_{f,j}]$ is the range of outputs of f corresponding to input points contained in H_j;
- $\mathbf{D} = (\mathbf{X}, \mathbf{Y})$ is a dataset where $\mathbf{X} = \{x_l\}_{l=1,\dots,L}$ is a list of inputs and $\mathbf{Y} = \{y_l\}_{l=1,\dots,L}$ is the list of corresponding outputs (labels), i.e., $\forall l : 1 \leq l \leq L, y_l = f(x_l)$;
- $e(x) = |f(x) - \mathcal{N}(x)|$ is the error made by $\mathcal{N}$ on input $x \in I$;
- $\tau \in \mathbb{R}$ is the error tolerance of $\mathcal{N}$.

3.1 Prerequisites and Assumptions

Prerequisite 1 *The ODD of the NN can be represented as a hyperrectangle* $I = \times_{i=1,\dots,N}[l_i, u_i]$.

This is a reasonable prerequisite for NNs that model processes grounded in physical principles. The lower and upper bounds (l_i, u_i) on each input dimension can be derived from sensor saturations or from the limitations due to physics (e.g., an aircraft has a limited set of operating conditions, including limited range for speed and altitude).

Prerequisite 2 *The dataset* $\mathbf{D}$ *is such that the inputs* $\mathbf{X}$ *constitute a grid covering the entire ODD. Formally*

- *Each input dimension* i *is sampled with* k_i *values* $S_i = \{x_{i,1}, \ldots, x_{i,k_i}\}$ *where* $x_{i,1} = l_i \leq x_{i,2} \leq \ldots \leq x_{i,k_i} = u_i$*;*
- *The list of inputs* $\mathbf{X}$ *is the combination of all the possible sampled values along every input dimension:*

$$\mathbf{X} = \{(x_{1,l} \ \ldots \ x_{N,l}) \mid \forall i = 1, \ldots, N, x_{i,l} \in S_i\} \tag{1}$$

Same as Prerequisite 1, this is reasonable when the NN is used to model a physical process. The ground truth f is often computable for every input in the grid $\mathbf{X}$, e.g., through a physics-based simulation model. Moreover, the sampled values along input dimensions can be obtained with sensitivity analysis.

Informally, Prerequisites 1 and 2 prescribe that the input domain of $\mathcal{N}$ is well defined, and the admissible range of each feature can be explicitly defined. Additionally, we specify the following assumption:

Assumption 1 *The ground truth function* f *has a known variation in between the points of the dataset* $\mathbf{D}$*. For instance,* f *can monotonic or* K*-Lipschitz between the points of* $\mathbf{D}$*, i.e., its slope can be bounded by a constant* $K \in \mathbb{R}$*.*

3.2 Property Formalization

In our approach, NN generalization capability is formalized by considering the behavior of the NN on *any* possible input, i.e., any input within its the ODD. More specifically, we consider a property $\mathcal{P}$ which states that the error made by $\mathcal{N}$ on any input x shall be less than a given tolerance τ:

$$\mathcal{P} : \forall x \in I, \quad e(x) = |\mathcal{N}(x) - f(x)| \leq \tau \tag{2}$$

The property $\mathcal{P}$ states that the worst case error of $\mathcal{N}$ on any input within its ODD I shall not exceed the tolerance τ. This tolerance represents a bound on the out-of-sample error of the NN and can be allocated from NN performance requirements.

$\mathcal{P}$ is a *global* property because it is imposed for the entire ODD of the NN. To facilitate its verification, we break it down into a collection of *local* properties. This formalization relies on:

- A partition of the ODD into a set of hyperrectangles $\mathbf{H} = (H_1, \ldots, H_Q)$;
- A set of local properties $\mathcal{P}_1, \ldots, \mathcal{P}_Q$, where the property $\mathcal{P}_j$ states that the error made by $\mathcal{N}$ in the hyperrectangle H_j shall not exceed τ:

$$\mathcal{P}_j : \forall x \in H_j, \quad e(x) = |\mathcal{N}(x) - f(x)| \leq \tau \tag{3}$$

Since the set $\mathbf{H}$ constitutes a partition of the ODD, then $\mathcal{P}_1 \wedge \ldots \wedge \mathcal{P}_Q \Rightarrow \mathcal{P}$.

Remark 1. The formalization above can be extended to the case where $\mathcal{N}$ has $M > 1$ outputs. A possible formulation is to specify a dedicated error tolerance for each NN output. The formulation of $\mathcal{P}$ then becomes

$$\mathcal{P}' : \forall m = 1, \ldots, M,\ e(x)_m = |f(x)_m - \mathcal{N}(x)_m| \leq \tau_m \tag{4}$$

where $e(x)_m$ is the error made by $\mathcal{N}$ on its m^{th} output and τ_m is the error tolerance for the m^{th} output of $\mathcal{N}$.

3.3 Verification Algorithm

To verify the properties formalized above, we propose an algorithm that consists of four main steps, which are illustrated in Fig. 1 and described below.

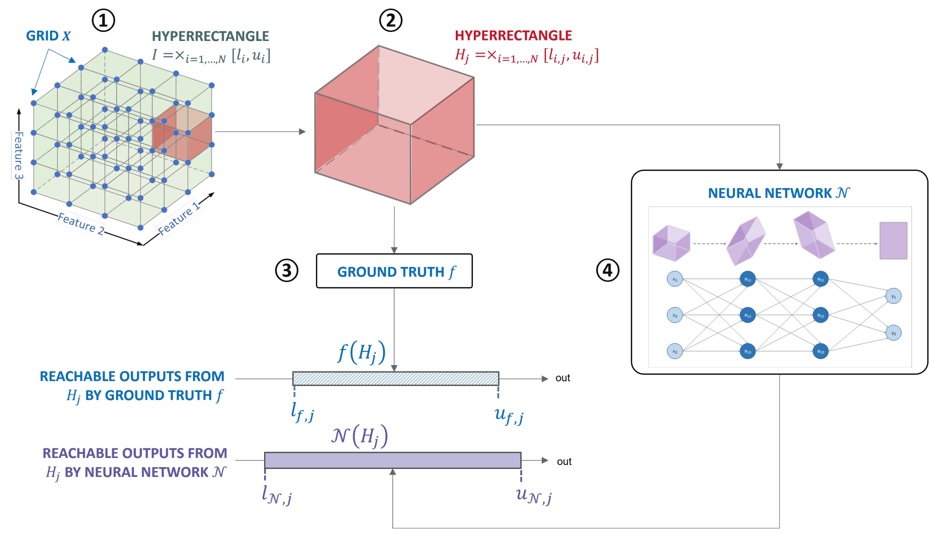

①
GRID X
HYPERRECTANGLE
$I = \times_{i=1,\ldots,N} [l_i, u_i]$
Feature 3
Feature 2
Feature 1
②
HYPERRECTANGLE
$H_j = \times_{i=1,\ldots,N} [l_{i,j}, u_{i,j}]$
NEURAL NETWORK $\mathcal{N}$
③
GROUND TRUTH f
④
$f(H_j)$
REACHABLE OUTPUTS FROM H_j BY GROUND TRUTH f
out
$l_{f,j}$
$u_{f,j}$
$\mathcal{N}(H_j)$
REACHABLE OUTPUTS FROM H_j BY NEURAL NETWORK $\mathcal{N}$
out
$l_{\mathcal{N},j}$
$u_{\mathcal{N},j}$

Figure 1: Formal verification algorithm for NN generalization assessment.

Step 1 (Partitioning) The first step is the partitioning of the ODD in a set of hyperrectangles $\mathbf{H} = (H_1, ..., H_Q)$, the corners of which are the points $\mathbf{X} = \{x_l\}_{l=1,\ldots,L}$ of the dataset $\mathbf{D}$. This step is made possible given that the ODD can be represented by a hyperrectangle I (see Prerequisite 1) and that the inputs $\mathbf{X}$ form a grid which covers the entire set I (see Prerequisite 2).

Step 2 (Hyperrectangle retrieval) The second step consists in retrieving the bounds $(l_{i,j}, u_{i,j})$ of a hyperrectangle H_j from $\mathbf{H}$.

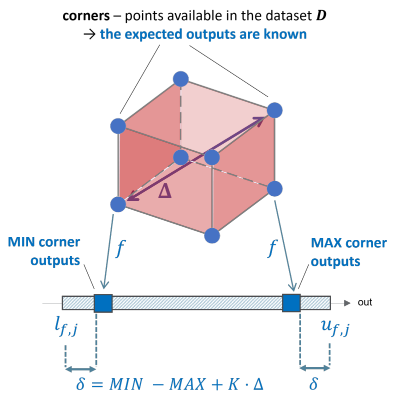

corners – points available in the dataset ***D***
→ **the expected outputs are known**
Δ
MIN corner outputs
MAX corner outputs
f
f
out
$l_{f,j}$
$u_{f,j}$
$\delta = MIN - MAX + K \cdot \Delta$
δ

Figure 2: Over-approximation of $f(H_j)$ when f is K-Lipschitz.

Step 3 (Computing the reachable outputs of f) This step computes a sound approximation $f(H_j)$ of the range of the reachable outputs of f from H_j by:

1. Retrieving the expected outputs at the n_j corners of the hyperrectangle H_j, denoted by $y_{1,j}, \ldots, y_{n_j,j}$, which are by definition available in the dataset $\mathbf{D}$;
2. Computing the range $f(H_j) = [l_{f,j}, u_{f,j}]$ by leveraging Assumption 1 i.e., the knowledge available on the variation of the ground truth f in between the corners of H_j. For instance:
   - If f is monotonic inside H_j, then exact bounds can be computed: $l_{f,j} = \mathrm{MIN} := \min_{l=1,\ldots,n_j}(y_{l,j})$ is the minimum values among of the expected outputs at the corners and $u_{f,j} = \mathrm{MAX} := \max_{l=1,\ldots,n_j}(y_{l,j})$ is the maximum value;
   - If f is K-Lipschitz inside H_j and K is known, then over-approximating bounds $(l_{f,j}, u_{f,j})$ can be computed (see Figure 2):

$$\begin{cases} l_{f,j} &= \mathrm{MIN} - \delta \\ u_{f,j} &= \mathrm{MAX} + \delta \end{cases} \tag{5}$$

   where $\delta = \mathrm{MIN} - \mathrm{MAX} + K \cdot \varDelta$ and $\varDelta$ is the maximum distance between two inputs in H_j i.e., $\varDelta = |(l_{1,j}, \ldots, l_{N,j}) - (u_{1,j}, \ldots, u_{N,j})|$, where $l_{i,j}$ is the lower bound of H_j along the i^{th} dimension and $u_{i,j}$ is the upper bound of H_j along the i^{th} dimension.

Step 4 (Computing the reachable outputs of $\mathcal{N}$) This step computes a sound approximation $\mathcal{N}(H_j)$ of the range of the reachable outputs of $\mathcal{N}$ from H_j. It can be computed, for example, based on abstract interpretation, i.e., by relying on an abstract transformer $\mathcal{N}^{\#}$ that soundly approximates the semantics of $\mathcal{N}$:

$$\mathcal{N}(H_j) = \mathcal{N}^{\#}(H_j) \tag{6}$$

Remark 2. If $\mathcal{N}$ involves pre-processing and post-processing functions, they also need to be handled by dedicated abstract transformers. Since pre-processing and post-processing are often linear transformations that do not introduce any dependency among input variables, they can be handled efficiently with a transformer based on interval abstract domain.

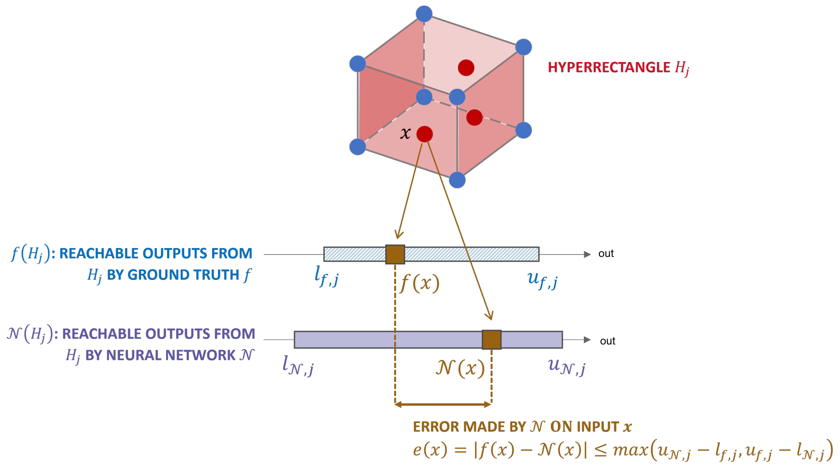

HYPERRECTANGLE H_j
x
$f(H_j)$: REACHABLE OUTPUTS FROM H_j BY GROUND TRUTH f
out
$l_{f,j}$
$f(x)$
$u_{f,j}$
$\mathcal{N}(H_j)$: REACHABLE OUTPUTS FROM H_j BY NEURAL NETWORK $\mathcal{N}$
out
$l_{\mathcal{N},j}$
$\mathcal{N}(x)$
$u_{\mathcal{N},j}$
ERROR MADE BY $\mathcal{N}$ ON INPUT x
$e(x) = |f(x) - \mathcal{N}(x)| \leq max(u_{\mathcal{N},j} - l_{f,j}, u_{f,j} - l_{\mathcal{N},j})$

Figure 3: Computation of an upper bound on the error made by $\mathcal{N}$ in the hyperrectangle H_j.

Once the range $\mathcal{N}(H_j)$ is computed, it is compared to the range $f(H_j)$ from Step 3 to determine an upper bound on the error made by the NN in the hyperrectangle H_j i.e., a bound $E_j \in \mathbb{R}$ such that:

$$\forall x \in H_j, e(x) = |f(x) - \mathcal{N}(x)| \leq E_j \tag{7}$$

where $e(x)$ is the error made by $\mathcal{N}$ on the input $x \in H_j$.

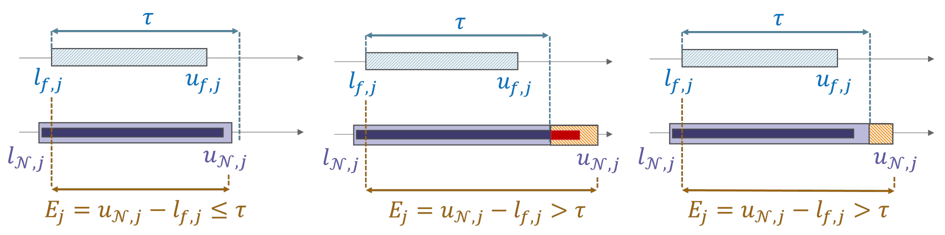
By definition, for all input x in H_j:

$$\begin{cases} f(x) \in f(H_j) = [l_{f,j}, u_{f,j}] \\ \mathcal{N}(x) \in \mathcal{N}(H_j) = [l_{\mathcal{N},j}, u_{\mathcal{N},j}] \end{cases} \tag{8}$$

Hence,

$$\begin{aligned} f(x) - \mathcal{N}(x) &\in [l_{f,j} - u_{\mathcal{N},j}, u_{f,j} - l_{\mathcal{N},j}] \\ \Rightarrow |f(x) - \mathcal{N}(x)| &\leq \underbrace{\max(|l_{f,j} - u_{\mathcal{N},j}|, |u_{f,j} - l_{\mathcal{N},j}|)}_{E_j} \end{aligned} \tag{9}$$

Finally, it can be checked if the bound on the error $E_j := \max(|l_{f,j} - u_{\mathcal{N},j}|, |u_{f,j} - l_{\mathcal{N},j}|)$ is less than the error tolerance τ. There are two possible outcomes:


τ
$l_{f,j}$
$u_{f,j}$
$l_{\mathcal{N},j}$
$u_{\mathcal{N},j}$
$E_j = u_{\mathcal{N},j} - l_{f,j} \leq \tau$
$E_j = u_{\mathcal{N},j} - l_{f,j} > \tau$
$E_j = u_{\mathcal{N},j} - l_{f,j} > \tau$

(a) Property valid (b) True negative (c) False negative

Figure 4: Examples of outcomes of Step 4 of the algorithm. The exact set of the reachable outputs of $\mathcal{N}$ is represented in dark purple for illustrative purposes.

- If $E_j \leq \tau$, the property $\mathcal{P}_j$ is proven *valid* i.e., the error made by $\mathcal{N}$ in H_j is effectively less than the error tolerance τ (see Figure 4a);
- If $E_j > \tau$, no conclusion can be drawn on the validity of the property $\mathcal{P}_j$ which remains *Unknown*. Indeed, two cases are possible:
  1. *True negative*: the property $\mathcal{P}_j$ is *invalid* and there exist a counterexample i.e., an input $x^* \in H_j$ such that $e(x^*) > \tau$ (see Figure 4b);
  2. *False negative*: the property $\mathcal{P}_j$ is *valid*, yet, due to the approximation of the reachable outputs of either f or $\mathcal{N}$, the bound E_j on the error is too conservative and does not allow to prove $\mathcal{P}_j$ valid (see Figure 4c).

In the latter case where $E_j > \tau$, to decide whether this is a true negative or a false negative, a simple counterexample search can be performed. The error can be computed for every corner of the hyperrectangle H_j, for which the expected outputs $y_{1,j}, \ldots, y_{n_j,j}$ are available in the dataset. If there exists a corner where the error is greater than τ then this is a counterexample which disproves that the property $\mathcal{P}_j$ holds. Otherwise, a refinement of the hyperrectangle H_j can be performed, as explained in Section 4.

Steps 2, 3 and 4 are repeated until all the hyperrectangles composing the set $\mathbf{H}$ are analyzed. Once the analysis is complete, one can:

- Determine the ratio of the volume of the ODD where the generalization property is valid;
- Record the list $\mathbf{H}_{unknown}$ of all the hyperrectangles of $\mathbf{H}$ where the validity of the property $\mathcal{P}_j$ remains unknown.

The latter output can be used to design a monitor that during operation can detect whether the incoming input belongs to one of the unproven hyperrectangles, in which case the NN is not activated (e.g., a backup algorithm can be used instead or a warning can be shown). This can ensure a safe usage of the NN.

Remark 3. While we employ abstract interpretation to compute NN reachable outputs at Step 4, other approaches can also be used, such as SMT solvers, if the NN has piecewise linear activation functions. One can express the existence of a counterexample as the existence of an x in H_j such that the following disjunction holds:

$$(\mathcal{N}(x) - l_{f,j} \geq \tau) \vee (l_{f,j} - \mathcal{N}(x) \geq \tau) \vee (u_{f,j} - \mathcal{N}(x) \geq \tau) \vee (\mathcal{N}(x) - u_{f,j} \geq \tau)$$

The four clauses in this disjunction can be checked with SMT. If none of them is satisfiable, i.e., there exists no x such that they are satisfied, then there exists no counterexample and the property $\mathcal{P}_j$ can be concluded valid.

3.4 Implementation and Optimizations

The verification method is currently implemented in MATLAB and the abstract transformer $\mathcal{N}^{\#}$ of $\mathcal{N}$ relies on the MATLAB Deep Learning Toolbox Verification Library [1] which implements the DeepPoly approach [22]. This approach leverages a dedicated, polytope-based abstract domain for the analysis of NNs. It offers a good balance between accuracy (i.e., tightness of the bounds of the range $\mathcal{N}(H_j)$ of the reachable outputs of $\mathcal{N}$) and cost of the analysis (time to compute the range $\mathcal{N}(H_j)$ with $\mathcal{N}^{\#}$). In addition, *vectorization* and *sorting and indexing*, have been implemented to accelerate the execution of the method. These optimizations are described below.

Vectorization Current hardware and software provide direct support for vector operations where a single instruction is applied to multiple data. This offers a faster execution time compared to applying the instruction to a single data. Therefore, Steps 2, 3 and 4 of the method are vectorized such that the corresponding operations are applied not to a single hyperrectangle but to a *batch* of hyperrectangles: Step 2 retrieves a batch $H_{j_1}, \ldots, H_{j_B}$ of B hyperrectangles. Then Steps 3 and 4 are applied to these B hyperrectangles to assess the validity of the corresponding batch of properties $\mathcal{P}_{j_1}, \ldots, \mathcal{P}_{j_B}$.

Sorting and Indexing When the dataset $\mathbf{D}$ is large, searching $\mathbf{D}$ for the expected outputs at the corners of a specific hyperrectangle (as needed in Step 3) is a costly operation, which may take significant time compared to the computation of the reachable outputs of the NN with abstract interpretation. To mitigate this, the dataset $\mathbf{D}$ can be sorted and the set $\mathbf{H}$ can be indexed. In our implementation, the dataset is sorted based on the values of the inputs: $\mathbf{X}$ is sorted in ascending order of the value of the first input dimension, then where the value of the first input dimension is identical, $\mathbf{X}$ is sorted in ascending order of the value of the second input dimension, and this is repeated until last input dimension. The list $\mathbf{Y}$ is then sorted such that its l^{th} element is the expected output for the l^{th} input in the sorted list $\mathbf{X}$. The indexing of the set of hyperrectangles is done by considering that $\mathbf{H}$ is ordered in ascending order of the lower bound of the first input dimension, then where the lower bound of the first input dimension is identical, $\mathbf{H}$ is ordered in ascending order of the lower bound of the second input dimension, and this is repeated until last input dimension:

$$\begin{aligned} H_1 &= [x_{1,1}, x_{1,2}] \times \ldots \times [x_{N,1}, x_{N,2}] \\ H_2 &= [x_{1,1}, x_{1,2}] \times \ldots \times [x_{N,2}, x_{N,3}] \\ H_3 &= [x_{1,1}, x_{1,2}] \times \ldots \times [x_{N,3}, x_{N,4}] \\ &\vdots \\ H_Q &= [x_{1,k_1-1}, x_{1,k_1}] \times \ldots \times [x_{N,k_N-1}, x_{N,k_N}] \end{aligned} \tag{10}$$

where, as indicated in Prerequisite 2, the values $x_{i,1}, \ldots, x_{i,k_i}$ are the sampled values along the i^{th} input dimension, defining the inputs in $\mathbf{X}$ and thus the corners of the hyperrectangles. The hyperrectangle H_j of index j is the j^{th} hyperrectangle in this ordered set. Considering this indexing of $\mathbf{H}$ and the sorting of the dataset $\mathbf{D}$:

- At Step 1, the set $\mathbf{H}$ needs not to be stored: it can be represented simply with the list $(Q_i)_{i=1,\ldots,N}$ where Q_i is the number of hyperrectangles along the i^{th} input dimension i.e., $Q_i = k_i - 1$, and the total number of hyperrectangles $Q = \prod_{i=1,\ldots,N} Q_i$. The set $\mathbf{H}$ can be explored by iterating over the index j between 1 and Q;
- At Step 2, the bounds of the hyperrectangle H_j of index j can be determined based on the sampled values along each inputs dimension and an iterative euclidean division of the index j by the values Q_i;
- At Step 3, similarly as the retrieving of the bounds of H_j, the expected outputs $y_{1,j}, \ldots, y_{n_j,j}$ at the corners of H_j can be retrieved by determining their indices in the sorted list $\mathbf{Y}$ with iterative euclidean divisions, alleviating the cost of searching $\mathbf{Y}$ for the combination of the N input values defining each corner of H_j.

3.5 Complexity

The computational complexity of the verification method was computed, based on the following considerations:

- The number of hyperretangles to be analyzed is in $O(L)$ where L is the size of the dataset $\mathbf{D}$. Indeed, $L = \prod_{i=1,\ldots,N} k_i$ where k_i is the number of sampled values along the i^{th} input dimension and the number of hyperrectangles is $Q = \prod_{i=1,\ldots,N} k_i - 1$;
- With sorting the data and indexing the hyperrectangle, the retrieval of the N bounds of a hyperrectangle H_j from $\mathbf{H}$ is in $O(N)$ where N is the number of input dimensions. Moreover, the retrieval of the expected outputs at the corners of a hyperrectangle is in $O(2^N)$ i.e., the number of corners of the hyperrectangle;
- The complexity of the computation of the reachable outputs of $\mathcal{N}$ from a hyperrectangle H_j with the DeepPoly approach is in $O(n_{layers}^2 \cdot n_{neurons}^3)$ where n_{layers} is the number of layers of $\mathcal{N}$ and $n_{neurons}$ is the maximum number of neurons in one layer [22].

Consequently, as the verification consists in retrieving the bounds and expected outputs at the corners of the Q hyperrectangles, and apply the DeepPoly approach on each of them, the total complexity is in $O(L \cdot 2^N \cdot n_{layers}^2 \cdot n_{neurons}^3)$. The applicability of the method is thus limited by three factors being: (1) the size of the dataset, (2) the number of input dimension and (2) the size of the NN. In practice, the method is applicable for low dimensional input spaces (N in the order of 10) and low-complexity NNs (number of learnable parameters in the order of $100 - 1000$).

4 Verification Workflow

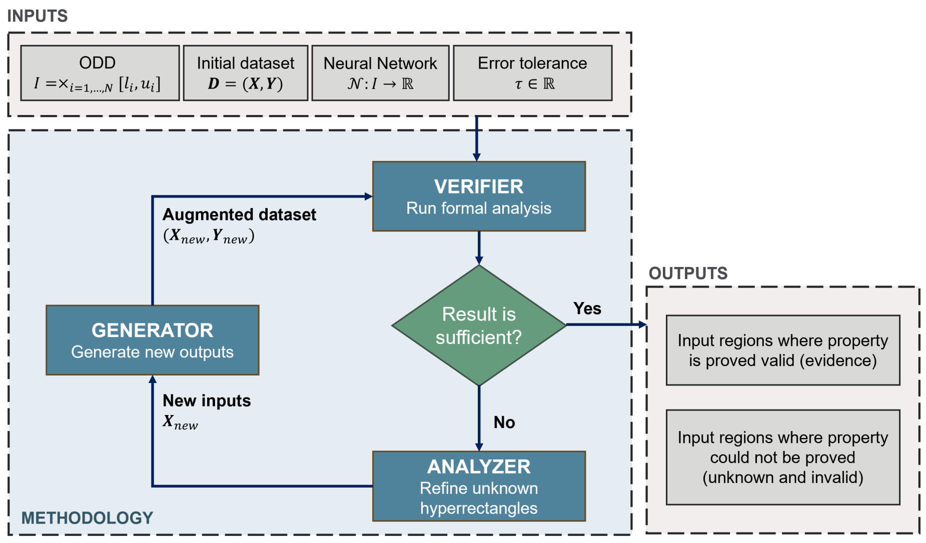

INPUTS
ODD
$I = \times_{i=1,\ldots,N} [l_i, u_i]$
Initial dataset
$\boldsymbol{D} = (\boldsymbol{X}, \boldsymbol{Y})$
Neural Network
$\mathcal{N} : I \to \mathbb{R}$
Error tolerance
$\tau \in \mathbb{R}$
VERIFIER
Run formal analysis
Augmented dataset
$(\boldsymbol{X}_{new}, \boldsymbol{Y}_{new})$
GENERATOR
Generate new outputs
Result is sufficient?
Yes
No
New inputs
$\boldsymbol{X}_{new}$
ANALYZER
Refine unknown hyperrectangles
METHODOLOGY
OUTPUTS
Input regions where property is proved valid (evidence)
Input regions where property could not be proved (unknown and invalid)

Figure 5: Iterative workflow with verification-driven refinement.

The possibility of proving the property (3) depends on the size of the hyperrectangle. Larger intervals lead to coarser approximations during verification and increase the risk of not being able to prove the property, i.e., to get an *Unknown* answer from the solver. A possible mitigation is to create a dataset with very high resolution, so that the hyperrectangles are "small enough". This may not be practical, for example, if running simulations to create data points are expensive or data generation is owned by a third party. Therefore, it is highly desirable to minimize the number of data points required to obtain the proof.

To this end, we propose a verification workflow that iteratively verifies the property and refines *only* those regions where the property could not be proven, which we call *verification-driven* data generation. This is implemented by a dedicated verification workflow, illustrated in Figure 5, which is composed of three main blocks: a *Verifier*, an *Analyzer* and a *Generator*. These three blocks are executed sequentially, as follows:

1. The Verifier implements the verification method described in Section 3, partitioning the ODD into a set $\mathbf{H}$ of hyperrectangles and checking the local property $\mathcal{P}_j$ in every of these hyperrectangles. It outputs the list $\mathbf{H}_{unknown}$ of the hyperrectangles where the validity of the property $\mathcal{P}_j$ remains unknown;
2. The Analyzer performs a *refinement* that splits the hyperrectangles pertaining to $\mathbf{H}_{unknown}$ into smaller hyperrectangles (see Figure 6). It outputs a list of new inputs $\mathbf{X}_{new}$, which are the corners of the smaller hyperrectangles, for which the expected outputs need to be computed to apply the verification method;
3. The Generator computes the outputs $\mathbf{Y}_{new} = f(\mathbf{X}_{new})$ for the corners of refined hyperrectangles, i.e., it is a labeling procedure.

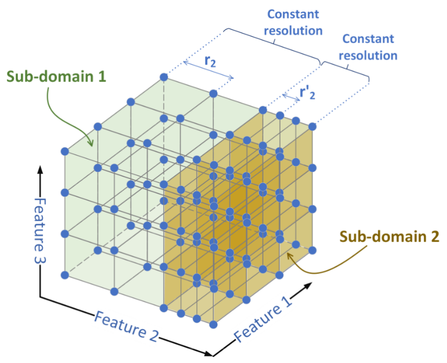


Figure 6: The Analyzer splits the hyperrectangles where the property is unknown, with a hyperrectangles of size r'_2 smaller than the original size r_2.

These three steps are repeated iteratively, until a termination criteria is reached. Such a criteria can be either:

1. The maximum resolution of inputs is reached;
2. Making further refinements is too costly;
3. The results are acceptable e.g., the volume of the regions where the property could not be proven is sufficiently small with respect to NN or system requirements.

An important hyperparameter of this workflow is the refinement performed by the Analyzer i.e., how every hyperrectangle dimension is split at each iteration. To determine this refinement, a hyperparameter tuning process is performed before running the verification workflow. This hyperparameter tuning relies on a heuristic that aims to find the refinement which maximizes the number of properties $\mathcal{P}_j$ proven valid after a given number of iterations, while minimizing the number of new data to be generated through the iterations. This heuristic consists in (1) defining a set of candidate refinements and (2) estimating the efficiency of these candidate refinements. For (1), a grid search approach can be followed e.g., consider the refinements which bisect all possible combinations of hyperrectangle dimensions. Expert knowledge about the ground truth or sensitivity analysis may also be used to identify the hyperrectangle dimensions to be split preferentially, and define a set of candidate refinements which split only these specific hyperrectangle dimensions. For (2), every candidate refinement is evaluated by running a modified version of the verification workflow, where the Generator is replaced with *linear interpolation*. The use of linear interpolation in this context allows estimating the efficiency of the candidate refinement i.e., the number of new data generated and the number of properties proven valid after a given number of iterations. Moreover, it avoids the cost of computing the ground truth function f for generating the new data.

5 Evaluation

5.1 Pilot Advisory System Case Study

The verification workflow presented in Section 4 is evaluated against an advisory system used onboard aircraft to provide the pilot with the optimal altitude. This advisory system takes as inputs a candidate optimal altitude as well as several variables describing the current state of the aircraft (weight, speed) and the weather conditions (wind, temperature). It outputs the cost of flying at the candidate altitude given the current state of the aircraft and the weather conditions.

This advisory system is implemented as a NN with ReLU activation functions and two hidden layers of 13 neurons each. This NN aims to approximate a traditional implementation of the advisory system while offering reduced execution time and memory footprint. Regarding generalization capabilities, it is expected that, for any input in the ODD, the relative error made by the NN is less than 5% compared to the traditional implementation. Formally, if $\mathcal{N}$ denotes the NN implementing the advisory system, f the traditional implementation and I the ODD, the desired property is:

$$\mathcal{P}_{5\%}: \ \forall x \in I, \ e(x) = |f(x) - \mathcal{N}(x)| \leq 5\% \cdot |f(x)| \tag{11}$$

where the tolerance on the error made by $\mathcal{N}$ on input x i.e., the quantity $5\% \cdot |f(x)|$, depends on the value of the input x. As one can note, this is slightly different from the property $\mathcal{P}$ addressed by our verification method, where the tolerance on the error is a fixed value $\tau \in \mathbb{R}$ (see Equation 2). This difference was handled by reformulating each local property $\mathcal{P}_j$ as:

$$\mathcal{P}_{j,5\%} : \ \forall x \in H_j, \ e(x) = |f(x) - \mathcal{N}(x)| \leq \tau_{j,5\%} \tag{12}$$

where τ_j is such that:

$$\forall x \in H_j, \ \tau_{j,5\%} \leq 0.05 \cdot |f(x)| \tag{13}$$

To satisfy this condition, $\tau_{j,5\%}$ was taken equal to 5% of the minimum of $|f(x)|$ in H_j i.e. $\tau_{j,5\%} := l_{f,j}$ where $l_{f,j}$ is the lower bound on the reachable outputs of f from H_j. Indeed, $\min_{x \in H_j} |f(x)| = \min_{x \in H_j} f(x) = l_{f,j}$ since $f(x)$ is a cost, assumed to take positive values only.

5.2 Experiments

The verification of the advisory system was performed with a dataset $\mathbf{D} = (\mathbf{X}, \mathbf{Y})$ containing 4 million points. As required by Prerequisite 2, this dataset $\mathbf{D}$ constitutes a grid which covers the entire ODD of the advisory system. In addition, the variation of the traditional implementation f is known (monotonic) in between the points of this dataset $\mathbf{D}$, satisfying Assumption 1.

Table 1: Results of the generalization assessment of the advisory system with 3 iterations of the verification workflow, run on Windows machine and NVIDIA RTX A6000 GPU with 48GB memory.

	Iteration 1	**Iteration 2**	**Iteration 3**
Number of Data	4M	66M	171M
Number of Hyperrectangles	2M	48M	164M
Ratio of volume proven valid	0%	59%	91%
Ratio of volume proven invalid	4%	2%	2%
Ratio of volume unknown	96%	39%	7%
Time elapsed[a]	3 minute	15 minutes	35 minutes

[a] This time includes only the time to run the Verifier and the Analyzer, excluding the time to run the Generator which computes the new outputs by executing the traditional implementation f.

The generalization assessment of the advisory system was performed with the following choice of hyperparameters:

- A number of 3 iterations of the verification workflow i.e., three executions of the Verifier, Analyzer and Generator;
- A batch size B of 1 million hyperrectangles for the vectorization of the operations involved in the Verifier and the Analyzer;

- A refinement (i.e., how the Analyzer refines the unknown hyperrectangles at each iteration) determined with the heuristic described in Section 4: several candidate refinements were defined, based on expert knowledge and sensitivity analysis, and tested by running 3 iterations of the workflow and using *linear interpolation* to generate the new data instead of the ground truth.

The results of the generalization assessment are given in Table 1, which shows, for each iteration of the workflow:

- Amount of data generated since first iteration. At iteration 1, this number is the size of dataset $\mathbf{D}$ while at iteration 2, it is the size of $\mathbf{D}$ plus the number of data generated to refine the unknown hyperrectangles $\mathbf{H}_{unknown}$ in iteration 1;
- Number of hyperrectangles analyzed since first iteration;
- Ratio of the volume of the Operational Design Domain I where the desired property $\mathcal{P}_{j,5\%}$ is proven valid i.e., the ratio between (i) the sum of the volumes of the hyperrectangles H_j where $\mathcal{P}_{j,5\%}$ is proven valid and (ii) the total volume of the ODD;
- Ratio of the volume of the ODD I where the desired property $\mathcal{P}_{j,5\%}$ is proven invalid (through counterexamples found at one corner of the invalid hyperrectangles);
- Ratio of the volume of the ODD I where the validity of the desired property $\mathcal{P}_{j,5\%}$ remains unknown;
- Time elapsed since the first iteration (excluding the time to run the Generator which computes new outputs with f).

It can be seen that after 3 iterations of the workflow, the desired property can be proven valid in 91% of the total volume of the ODD, requiring a total of 171 million data points to be generated and 35 minutes of verification time. The iterative refinements of the unknown hyperrectangles provide a significant increase of the volume where the property is proven valid, from 0% at iteration 1, due to the large size of the hyperrectangles defined by the initial dataset $\mathbf{D}$, to 59% at iteration 2 and 91% at iteration 3. In the remaining 9% of volume at iteration 3, 7% remains unknown and 2% is proven invalid. This ratio where the property is proven invalid decreases along the iterations. This is due to the fact that invalid hyperrectangles are also refined along iterations: only some of the refined hyperrectangles have invalid corners while the other refined hyperrectangles are not invalid, hence the decreasing ratio of invalid areas.

Comparison with Statistical Methods As mentioned in the related work, several statistical methods can also be applied to assess the property $\mathcal{P}_{j,5\%}$ which we verified formally with our workflow. These statistical methods were also applied to the advisory system [6]. To reach the desired confidence level, statistical methods required to generate 10^9 data, which is one order of magnitude above the amount of data generated in our verification, representing additional computational cost. Moreover, one advantage of the formal verification compared to statistical methods is the possibility to identify the hyperrectangles where the property cannot be proven valid. This information can be leveraged for instance to design a real-time monitoring mechanism, such as a safety net [7], which switches to the traditional implementation f when the input lies in one of these areas.

Evaluation of Optimizations The acceleration of the verification time due to optimizations discussed in Section 3.4 was also measured. It was observed that the vectorization of the operations with batches of 1 million hyperrectangles, parallelized on a 48GB memory GPU, provides an acceleration by a factor in the order of 10^4. Regarding the sorting and indexing of data and hyperrectangles, it was observed that it provides an acceleration by a factor in the order of 10^2 to 10^3, depending on the size of the dataset and the number of hyperrectangles.

6 Related Work

Model generalization is an important topic in theoretical machine learning, specifically in the field of statistical learning theory. The study of quantitative generalization bounds has a rich history that started with seminal works, such as VC-dimension and Rademacher complexity, and produced various forms of bounds since then [9]. Many bounds are based on the Probably Approximately Correct (PAC) Bayesian theory (e.g., see [17], [8]). Such bounds are often vacuous and thus impractical for deep learning applications [16], but can be computed sufficiently tight for low-complexity NNs, which are the focus of this work. Still, to achieve high confidence levels (e.g., 99.9% and beyond) that is required for assuring a safety-critical NN, very large sample may be needed, whereas producing additional data points may incur high costs due to various reasons (expensive to generate, data owned by a third party, etc.). In contrast to statistical methods, our generalization assessment approach is based on formal methods and provides mathematical guarantees on the validity of the generalization property (basically, a 100% confidence). We have also shown that this approach requires significantly less data points for the analysis.

The use of formal methods for learning assurance of safety-critical ML applications in aviation is discussed in [10] and [2]. The report provides seminal ideas for generalization assessment guarantees, but no practical solution, which is instead materialized in this paper. Additionally, a similar approach for dividing the NN input space into sub-regions (called "boxes") has been presented in [7]. However, their work focuses on verification of the ACAS-XU system, which is based on classification NNs. Our current result has been produced for regression NNs, however, the generalization property can be reformulated for classification networks as well, so that the method would apply to ACAS-XU and similar problems. This merely depends on how the tolerance is defined in the property. In [4], formal methods are used for NN generalization assessment: several NNs are trained independently and formal methods are used to identify the NNs which are in agreement across all inputs in a specific domain. This approach evaluates generalization by comparing several independently trained NNs, implying that the NNs which are in agreement have learned decision rules that generalize well to all inputs. However, our work focuses on verifying a NN against the ground truth and providing a proof that the error made by the NN, compared to the ground truth, is below a given tolerance.

Finally, it is worth noting that generalization capability is one of the key ML assurance objectives in the emerging aviation guidance, including [3], [21] and [18]. These documents may use different terminology (for example, the FAA does not explicitly speak about generalization, but rather refers to measuring expected empirical error and level of confidence that it corresponds to the true error), but the expected evidence is the same. Our approach can complement the anticipated means of compliance for generalization objectives with formal methods-based approaches that have a clear benefit of providing formal guarantees rather than statistical. Even though analyzing deep learning models remains a challenge for formal methods due to scalability barriers, this approach can extensively support low-complexity NNs that find many applications in safety-critical aviation systems.

7 Conclusion

We presented a mathematical formalization of the generalization property for NNs, and a method to prove this property with formal analysis. We also proposed a workflow that limits the amount of data required for obtaining the proof by combining the verification method with an iterative refinement of the regions where the property could not be proven valid. We demonstrated the applicability of the approach on a realistic case study of a safety-critical pilot advisory system that uses a low-complexity regression NN. Our results provided *formal* generalization guarantees for the significant portion of the NN input space and used orders of magnitude fewer data points compared to an existing statistical method that was employed to verify this NN. Future work will focus on enhancing the choice of the workflow hyperparameters, such as the choice of the optimal refinement strategy. We further plan to extend the workflow to support NN improvements (e.g., fine tuning). Finally, we also plan to assess the scalability of the approach on higher complexity NNs to identify further optimizations.

References

1. Deep learning toolbox verification library. https://fr.mathworks.com/products/deep-learning-verification-library.html, accessed: 2025-04-29
2. Machine Learning in Certified Systems. Tech. rep., DEEL Certification Working group (2020)
3. EASA Artificial Intelligence Concept Paper Issue 2: Guidance for Level 1&2 machine learning applications. Tech. rep., EASA (March 2024)
4. Amir, G., Maayan, O., Zelazny, T., Katz, G., Schapira, M.: Verifying Generalization in Deep Learning. In: Computer Aided Verification. pp. 438–455. Springer (2023). https://doi.org/10.1007/978-3-031-37703-7_21

5. Brix, C., Bak, S., Johnson, T.T., Wu, H.: The Fifth International Verification of Neural Networks Competition (VNN-COMP 2024): Summary and Results. arXiv preprint arXiv:2412.19985 (2024)
6. Collins Aerospace: Certification Aspects of Collins Aerospace Recommended Cruise Level Neural Network Development. Tech. rep. (2023)
7. Damour, M., De Grancey, F., Gabreau, C., Gauffriau, A., Ginestet, J.B., Hervieu, A., Huraux, T., Pagetti, C., Ponsolle, L., Clavière, A.: Towards certification of a reduced footprint ACAS-XU system: A hybrid ML-based solution. In: Proc. of SAFECOMP. pp. 34–48. Springer (2021). https://doi.org/10.1007/978-3-030-83903-1_3
8. Dziugaite, G.K., Roy, D.M.: Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. arXiv:1703.11008 (2017)
9. EASA AI Task Force and Daedalean, AG: Concepts of Design Assurance for Neural Networks (CoDANN). Tech. rep. (2020)
10. EASA and Collins Aerospace: Formal Methods use for Learning Assurance (ForMuLA). Tech. rep. (April 2023)
11. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: Automated Technology for Verification and Analysis (2017)
12. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: AI ^{2}: Safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 3–18 (2018). https://doi.org/10.1109/SP.2018.00058
13. Julian, K.D., Kochenderfer, M.J., Owen, M.P.: Deep neural network compression for aircraft collision avoidance systems. Journal of Guidance, Control, and Dynamics **42**(3), 598–608 (2019). https://doi.org/10.2514/1.G003724
14. Kirov, D., Rollini, S.F.: Benchmark: remaining useful life predictor for aircraft equipment. In: Proceedings of AISOLA. pp. 299–304. Springer (2023). https://doi.org/10.1007/978-3-031-46002-9_18
15. Kirov, D., Rollini, S.F., Chandrahas, R., Chandupatla, S.R., Sawant, R.: Benchmark: Object detection for maritime search and rescue. In: Proceedings of AISOLA. pp. 305–310. Springer (2023). https://doi.org/10.1007/978-3-031-46002-9_19
16. MLEAP Consortium: EASA Research – Machine Learning Application Approval (MLEAP) final report. HORIZON Europe research and innovation programme report, EASA (May 2024)
17. Pérez-Ortiz, M., Rivasplata, O., Shawe-Taylor, J., Szepesvári, C.: Tighter risk certificates for neural networks. Journal of Machine Learning Research **22**(227), 1–40 (2021)
18. Pham, T.: Technical concept paper 1: Verification of an artificial neural net model developed through machine learning. Tech. rep., FAA (2024)
19. RTCA/DO-178C: Software Considerations in Airborne Systems and Equipment Certification (2011)
20. SAE G-34 Artificial Intelligence in Aviation: Artificial Intelligence in Aeronautical Systems: Statement of Concerns (2021)
21. SAE G34 / EUROCAE WG-114: ARP6983 / ED-324: Process Standard for Development and Certification/Approval of Aeronautical Safety-Related Products Implementing AI (DRAFT 6b) (2024)
22. Singh, G., Gehr, T., Püschel, M., Vechev, M.: An abstract domain for certifying neural networks. Proc. ACM Program. Lang. **3**(POPL) (2019). https://doi.org/10.1145/3290354

23. Tjeng, V., Xiao, K.Y., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: International Conference on Learning Representations (2019)
24. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.S.: Formal security analysis of neural networks using symbolic intervals. In: USENIX Security Symposium. USENIX Association (2018)
25. Wu, H., Ozdemir, A., Zeljić, A., Julian, K., Irfan, A., Gopinath, D., Fouladi, S., Katz, G., Pasareanu, C., Barrett, C.: Parallelization techniques for verifying neural networks. In: 2020 Formal Methods in Computer Aided Design (FMCAD). pp. 128–137. IEEE (2020). https://doi.org/10.34727/2020/isbn.978-3-85448-042-6_20
26. Zhang, H., Weng, T.W., Chen, P.Y., Hsieh, C.J., Daniel, L.: Efficient neural network robustness certification with general activation functions. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. pp. 4944–4953. NIPS'18, Curran Associates Inc. (2018)