Formal Verification of a Neural Network Based Prognostics System for Aircraft Equipment

Dmitrii Kirov*, Simone Fulvio Rollini, Luigi Di Guglielmo, and Darren Cofer

Collins Aerospace

Abstract. We demonstrate the use of formal methods to verify properties of a deep convolutional neural network that estimates remaining useful life of aircraft mechanical equipment. We provide mathematical formalizations of requirements of the estimator, such as stability and monotonicity, as properties. To efficiently apply existing tools for verification of neural networks, we reduce the verification of global properties to a representative set of local properties defined for the points of the test dataset. We encode these properties as linear constraints and verify them using a state-of-the-art tool for neural network verification. To increase the completeness and the scalability of the analysis, we develop a two-step verification method involving abstract interpretation and simulation-based falsification. Numerical results confirm the applicability of the approach.

1 Introduction

The aviation industry is being increasingly driven towards the application of Machine Learning (ML) in new products to assist human operators or implement enhanced automation. Such products, in particular safety-critical ones, require certification and must provide a high level of trustworthiness and guarantees of the absence of unintended behaviors.

Currently, the industry does not have a consensus on the assurance of ML-enabled components because they are not fully amenable to current design assurance processes and standards. In particular, DO-178C provides guidance to produce traditional (i.e., non-ML) software that performs the intended function with a level of confidence in safety that complies with airworthiness requirements [12]. The standard focuses on a process for software design that starts from functional and non-functional requirements and transforms them into the software code. This code is traced to and verified against the requirements to ensure it is correct, i.e., it performs the intended function, and, more importantly, does not expose behaviors that are unintended by the designer or unexpected by operators.

ML development, instead, is data-driven. An ML model, such as a Neural Network (NN), is trained through a learning procedure that starts from data, not only from requirements. The use of traceability of the implementation back to

* Corresponding author. Email: dmitrii.kirov@collins.com

training data as a means to minimize the risk that the ML component includes unintended behaviors may not be possible. Additionally, the use of structural coverage metrics may not be effective in identifying unintended functionalities in ML models such as neural networks [14].

According to DO-333 [13], Formal Methods (FM) can be used as a source of evidence for the satisfaction of verification objectives when a formal model of the software artifact can be established and properties they have to comply with can be verified via formal analysis. FM provide a comprehensive analysis of a system over its entire input space, thus being able to show the absence of unintended behaviors. New FM tools are being developed for ML-based systems, in particular, for verification of neural networks (VNN) [7, 16, 15]. Several case studies on applying VNN tools have been published [9, 4]. Emerging certification guidance created by the European Union Aviation Safety Agency (EASA) explicitly mentions formal methods as promising means of compliance with a number of key assurance objectives, such as stability[1] and robustness of ML [6]. This is further elaborated in the recent technical report on the use of formal methods for learning assurance [5].

In this paper, we demonstrate the application of formal methods on an ML-enabled prognostics system for aircraft equipment to verify a function that predicts remaining useful life of mechanical components. The function is implemented as a convolutional neural network. The main contribution of the paper is to show the effectiveness of FM in identifying unintended behaviors of a real NN application that may potentially have an impact on safety. We were able to identify conditions, in which the requirements of the NN-based estimator, such as stability and monotonicity, are violated.

We also propose an approach to overcome scalability and applicability barriers of existing VNN tools for verifying global properties by reducing them to local properties defined for a representative set of input points. We discuss a necessary condition on data quality (completeness and representativeness) that enables such reduction. To further mitigate the scalability issue, we develop a hybrid, two-step verification method that involves abstract interpretation and simulation-based falsification.

2 Remaining Useful Life Estimator

Remaining Useful Life (RUL) is a metric that manifests the remaining lifetime of a component. In aviation RUL is used in Prognostics and Health Management (PHM) applications, such as condition-based maintenance, to support aircraft maintenance and flight preparation. RUL estimation could contribute to augmented manual inspection of components and scheduling of maintenance cycles. RUL could also highlight areas for inspection during the next planned maintenance, i.e., it could be used to move up (prioritize) a maintenance/inspection action to prevent component failure.

[1] This is a domain-specific concept, different e.g. from the notion of stability in control theory; see Section 3.1 for details.

Existing RUL calculation procedures often use physics-based degradation models [10]. Such models typically have high accuracy, but require extensive prior knowledge about the underlying physical system, which is often not available in practice. Similarity-based methods [2] are less complex to develop and deploy, but their main disadvantage is low accuracy. In the recent decade, the focus has been gradually shifting towards ML-based approaches [3, 11]. In particular, deep neural networks are capable of using raw sensor measurements directly as inputs, and leverage their automatic feature extraction capabilities to discover relationships between the inputs and their impact on the RUL, which may be unknown to the experts [8]. Therefore, they require less domain expertise and prior knowledge of the behavior of the equipment that is being monitored.

The RUL estimator examined in this case study is used in an on-ground maintenance application to provide information about the current state of a mechanical bearing component installed in the drivetrain assembly of a rotorcraft. End users are pilots, as well as Maintenance, Repair and Overhaul (MRO) teams. Predicted RUL is used during pre-flight checks to detect possible problems with the bearing and, if present, to prioritize certain maintenance/inspection actions.

The interface of the RUL estimator is illustrated on Fig. 1. The ML model is a Convolutional Neural Network (CNN) that accepts as input a time window of length 40, i.e., snapshots reflecting the bearing state taken at 40 consecutive time steps. Each time step corresponds to 1 hour. Each snapshot contains the following information:

- Seven Condition Indicators (CIs) that provide numerical information about the bearing degradation, obtained from signal processing of measurements of the vibration sensor attached to the bearing.
- Information about the current mission: current flight regime, such as ascent, cruise and descent, with corresponding nominal component load (bearing RUL heavily depends on how the component is loaded), mission type (a set of predefined mission patterns is available, e.g., long, short, mixed).
- Information about the current flight environment (e.g., desert or non-desert).

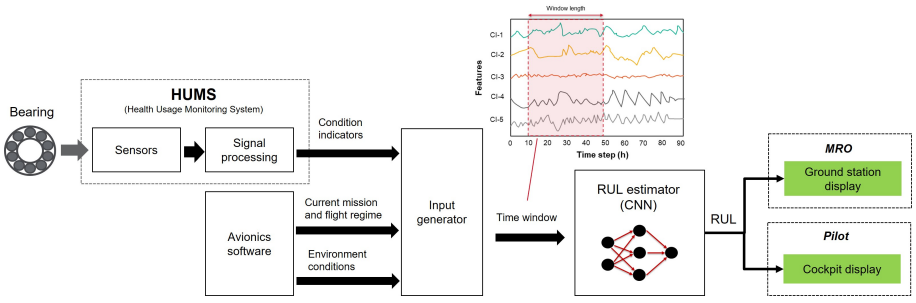

Fig. 1. Overview of the RUL estimator and its context.

CI values are computed by the Health Usage Monitoring System (HUMS), while remaining inputs are provided by other avionics software. Snapshots are provided to the input generator, where they are stored in memory and periodically shifted to yield a new consecutive time window. Output of the CNN is a numerical non-negative value that represents the bearing RUL in hours. It is provided to the end users (pilots, MRO) via cockpit and ground station displays. Additional pre- and post-processing of the CNN (e.g., input normalization) is not described due to space limitations, details are available in [5].

The NN architecture of the RUL estimator is adapted from [1]. It includes several convolutional layers that apply 1D convolutions along the time sequence direction of the input time window, thus extracting trends in separate features. These trends are merged together via a fully connected layer. Activation functions at all layers are Rectified Linear Units (ReLUs). The total number of layers is 12 and the number of learnable parameters is on the order of 100K. The CNN performs a regression task.

Requirements We provide a selected list of requirements of the RUL estimator in Table 1. They include stability and monotonicity of the estimator. These requirements shall be met within the entire Operational Design Domain (ODD) of the RUL estimator, i.e., the range of inputs for which the estimator is guaranteed to operate as intended. Violation of the requirements may have an impact on the safety, for example, if the NN overestimates the RUL. This is particularly relevant to the so-called *critical range* defined as the last **100h** of the bearing RUL, i.e., when the bearing may soon develop a failure. Overestimating the RUL in the critical range may lead to missing a critical component inspection before proceeding with the flight mission. Requirements from Table 1 can be formalized as properties, as will be shown in Section 3.

3 Property Formalization

In the remainder of this paper, the following notation is used. Bold font (e.g., $\mathbf{x}$) is used to denote a vector or a matrix, depending on the context, while normal font (e.g., x) denotes a scalar. We recall that a single input point for the RUL estimator function corresponds to an $L \times n$ time window with a predefined number of time steps L and n features ($1 \leq i \leq n$). Formally, an input point is a matrix $\boldsymbol{x} = [\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}]$, where each column i contains values x_i^t of some input feature i at consecutive time steps t ($1 \leq t \leq L$), i.e., a vector $\boldsymbol{x_i} = [x_i^1, \ldots, x_i^L]^T$.

Also, in the following, when considering two input points $\mathbf{x}$ and $\mathbf{x}'$, some step t^*, and a subset S of indexes of features, the values of the features not explicitly mentioned (i.e., their indexes not belonging to S) are assumed equal, i.e., $x_j^t = x_j^{'t}$ for $j \notin S$, and the values of the features in S are assumed equal at steps different from t^*, i.e., $x_j^t = x_j^{'t}$ for $j \in S, t \neq t^*$.

The scope of properties for the RUL estimator neural network can be either local or global. A *local property* is defined for a given input point $\boldsymbol{x} \in X$ or a

Table 1. Selected requirements for the RUL estimator.

ID	Requirement
RUL-ML-Stab-1	The maximum admissible perturbation that can occur to a condition indicator input shall be equal to **40%** of the average initial value of that CI that corresponds to a fully healthy state of the bearing component.
RUL-ML-Stab-2	For a perturbation of a single CI at a single time step within any input time window, the output deviation of the RUL estimator shall not exceed **10 hours**. The max perturbation for which the requirement must hold corresponds to RUL-ML-Stab-1. Requirement applies to each CI.
RUL-ML-Stab-3	For a simultaneous perturbation of all CIs at a single time step (e.g., due to a resonance frequency) within any time window, the output deviation of the estimator shall not exceed **10 hours**. The max perturbation for which the requirement must hold corresponds to RUL-ML-Stab-1.
RUL-ML-Mon-1	For an increased growth rate of a single CI (may occur, for example, when a particular failure occurs in the bearing, which increases its degradation) within any input time window, the estimator shall output a non-increasing value of the RUL. Requirement applies to each CI.
RUL-ML-Mon-2	For an increased growth rate of all CIs (may occur, for example, due to simultaneous development of a number of failures or due to excessive load) within any input time window, the estimator shall output a non-increasing value of the RUL.

subset of points $X' \subseteq X$ of the input space X. That is, local properties must hold for some specific inputs. Such properties are accepted by the majority of existing VNN tools. A *global property* is defined over the entire input space X of the NN. Global properties must hold for all inputs.

3.1 Stability Properties

The ML model is *stable* if a small, bounded perturbation applied to its inputs in normal operating conditions, i.e., when the inputs are inside its operational design domain, does not cause a significant deviation in its output [5]. Condition indicators are key input features of the RUL estimator for which undetected perturbations may occur during operation. For example, a spike change of a single CI may occur at some time step due to sensor noise. Similarly, a resonance frequency in the bearing may result in simultaneous deviation of multiple CIs (also at a single time step). Random CI perturbations at multiple, especially consecutive, time steps shall be detected by data quality monitors within the HUMS before they could enter the neural network, therefore, such cases are considered unrealistic. Similarly, spikes in CIs that exceed the admissible threshold prescribed by requirement RUL-ML-Stab-1 shall be detected by the HUMS.

Stability to a single CI perturbation at a single time step. Let $\boldsymbol{x_i}$ be the vector of values of some CI i in the time window, x_i^t being the CI value at time step t. The formulation of a stability property for a single CI is:

$$\forall \boldsymbol{x'} : |x_i^{'t} - x_i^t| < \delta |x_i^*| \rightarrow |f(\boldsymbol{x'}) - f(\boldsymbol{x})| < \epsilon, \tag{1}$$

where prime ($'$) denotes a perturbed item, i.e., $\boldsymbol{x'}$ is the time window, where one or more elements have been perturbed, $x_i^{'t}$ is a value of the i-th CI (perturbed) at time step t, and $f(\boldsymbol{x'})$ is the ML model output computed from the perturbed input, x_i^* represents the average initial value of the i-th CI, which can be computed by averaging initial CI values over all degradation scenarios in the available dataset, δ is the bound on the input perturbation *w.r.t.* x_i^* (expressed as a percentage), and ϵ is the maximum admissible output change.

Stability to multiple CI perturbations at a single time step. In case of perturbations over multiple CIs, the formulation of the stability property is:

$$\forall \boldsymbol{x'} : \forall i \in S : |x_i^{'t} - x_i^t| < \delta_i |x_i^*| \rightarrow |f(\boldsymbol{x'}) - f(\boldsymbol{x})| < \epsilon, \tag{2}$$

where S is a subset of the indexes corresponding to perturbed CIs, x_i^*, δ and ϵ are as above (the only difference is that for each perturbed CI with index i a different bound δ_i can be specified), and prime ($'$) denotes a perturbed time window ($\boldsymbol{x'}$) or value ($x_i^{'t}$).

3.2 Monotonicity Properties

Condition indicators and their growth rate are correlated with component degradation and failures. Differently from stability, requirements on monotonicity of the RUL estimator consider simultaneous systematic modifications of CIs at all time steps of the input window. This is a realistic situation that may occur, for example, due to damage or excessive load in the bearing that leads to an increased degradation rate. Therefore, such changes in the CIs over the entire time window shall not be identified by data quality indicators in the HUMS as abnormal (unlike random spikes/perturbations occurring at multiple time steps).

Expected behavior of the RUL estimator output is to be monotonic with respect to CIs, i.e., when they increase, the RUL should decrease. Faster bearing degradation is manifested by the CIs (one or many) growing faster. Monotonicity requirements in Table 1 prescribe a non-increasing output of the RUL estimator given an increase in the growth rate of one or more condition indicators.

Requirements RUL-ML-Mon-1 and RUL-ML-Mon-2 do not prescribe any concrete upper bound of the CI growth rate within any time window. In general, a growth rate increase of a CI by some percentage represents a different CI "trajectory" within the time window, as exemplified in Fig. 2a. Such new trajectory can be used as an upper bound, while the original CI growth trend represents a lower bound. The verification strategy would be to analyze all CI trajectories contained within these bounds, thus exhaustively verifying all possible CI changes within the interval and their effect on the monotonicity of the estimator.

A CI slope change affects all CI values in the time window. In particular, the values closer to the end of the window have a larger change. As a result, overall intervals to be analyzed by formal verification would become substantially

large, even if the upper bound (increased CI slope) is small. This may lead to scalability issues. Additionally, such formulation is not able to discard "oscillating" CI trajectories, i.e., highly non-monotonic[2] ones, since it only defines lower and upper bounds for CI values at each time step, but does not prescribe any interdependency between consecutive values. See example in Figure 2a (black line). Such CI trajectories may represent excessive noise but in general are not realistic. At the same time, they may invalidate a large number of properties, hampering the effectiveness of the verification. Hence, the analysis space has to be further restricted.

A compromise solution that enables a tradeoff between input space coverage, scalability, and effectiveness of the verification is presented below. It consists in extending the CI growth rate change with the possibility of a *constrained fluctuation*: this guarantees that, where the original CI trajectory is locally monotonically increasing, any trajectory in the defined interval is also monotonically increasing; where the original trajectory is not locally increasing, any derived trajectory is at least not "less monotonic" (see example in Figure 2b). Such formulation allows local oscillations in the CIs, for example, due to noise. Given a CI i, step t, constant γ, let $u_i^t = x_i^t + \gamma|x_i^t - x_i^1|$ and $v_i^t = x_i^t - \gamma|x_i^t - x_i^1|$. Monotonicity properties with constrained fluctuation for a single CI growth rate change can be expressed as:

$$\forall \boldsymbol{x'} : \forall t : u_i^t \leq x_i^{'t} \leq max(u_i^t, u_i^{t+1}) \rightarrow f(\boldsymbol{x'}) \leq f(\boldsymbol{x}) \tag{3a}$$

$$\forall \boldsymbol{x'} : \forall t : v_i^t \leq x_i^{'t} \leq max(v_i^t, v_i^{t+1}) \rightarrow f(\boldsymbol{x'}) \geq f(\boldsymbol{x}) \tag{3b}$$

Consider the Equation 3a. The difference $|x_i^t - x_i^1|$ represents an approximation of the CI slope in the interval $[1, t]$. The given CI i is allowed to fluctuate, at step t, between $u_i^t = x_i^t + \gamma|x_i^t - x_i^1|$ and $u_i^{t+1} = x_i^{t+1} + \gamma|x_i^{t+1} - x_i^1|$, if $u_i^{t+1} \geq u_i^t$, otherwise it is set to u_i^t. The *max* function captures this constrained fluctuation. Accordingly, at step $t+1$, x_i' either belongs to the interval $[u_i^{t+1}, u_i^{t+2}]$, or is set to u_i^{t+1}, if such interval is empty. This procedure guarantees that, if $x_i^t \leq x_i^{t+1}$, then $x_i^{'t} \leq x_i^{'t+1}$, leading to a derived trajectory that is not "less monotonic" than the original one. Similarly, Equation 3b captures the constrained CI fluctuation in the opposite direction, i.e., it prescribes a non-decreasing RUL in cases of a "slower" CI growth rate. Properties 3a-3b can be generalized to the case that captures simultaneous growth rate change in all CIs. These formulations are available in [5].

4 Verification Methods

We have implemented a formal verification framework to support the analysis of local stability and local monotonicity properties. The framework carries out all

[2] In this case, the notion of monotonicity applies to a CI in a time window, intended as a sequence of values, rather than to the RUL *w.r.t.* one of the CIs.

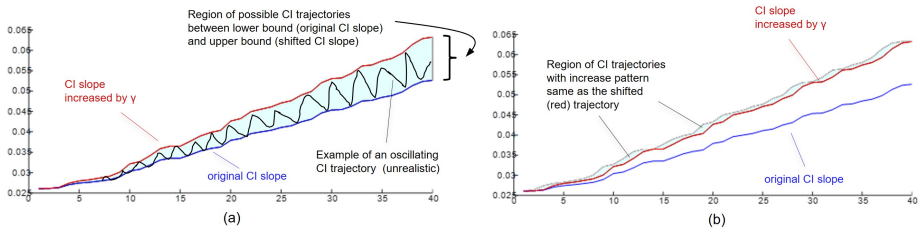

Fig. 2. Monotonicity analysis: (a) Entire space of possible CI trajectories bounded by the original CI trajectory at a given input point and a modified trajectory with an increased growth rate; (b) Subspace of CI trajectories with constrained fluctuation.

property formalization activities, i.e., it automatically creates property objects from numerical values provided in the requirements. To verify the properties, it then invokes the FM tool NNV [15] that is based on abstract interpretation. The tool computes the set of possible outputs for a set of inputs specified by the property, and then performs a geometrical check: for the property to be valid, the output set of the NN must not intersect with the region of the output space (halfspace) that is associated with the negation of this property (i.e., the "unsafe" area). Otherwise, an intersection manifests a property violation. The following verification methods are provided by the VNN tool:

- **Exact method.** This method performs exact reachability analysis for the neural network, i.e., it precisely computes the set of possible outputs (output reachable set) based on the provided input set.
- **Approximate method.** This method computes a conservative approximation of possible NN outputs via abstract interpretation. This enables faster analysis time. If the region associated with a property negation (i.e., unsafe region) does not intersect with the output set over-approximation, the property can be concluded valid. However, the analysis may not be able to disprove the property, that is, if an intersection with the negated property is found, the related counterexample may be spurious. To avoid raising a false alarm, in such situations the tool returns an "unknown" answer.

Despite being sound and complete, the exact verification method may face scalability issues when the complexity of the NN (e.g., number of layers, parameters) and/or the property (e.g., size of the input space to be considered) is high. On the other hand, the approximate method can only identify valid properties, but not invalid ones, always returning "unknown" in the latter case. In this paper, we propose a tradeoff between completeness and execution time. The new approach, called the **two-step method**, combines approximate verification described above, and a simulation-based *falsification* method. The latter randomly generates a number of inputs in the neighborhood of the given point for which the local property is defined, and performs NN inference to check the outputs against the property. If a violation is observed then the property is falsified and

can be declared invalid. However, contrary to the above methods, simulation alone cannot prove the property, it may only disprove it.

The two-step method is illustrated in Fig. 3. First, the property is verified with the approximate method and, if it is proven valid, the method terminates. Otherwise, if "unknown" is returned by the solver, the simulation-based method is invoked in an attempt to disprove the property by finding a counterexample (CEX) among a configurable number of randomly generated inputs in the neighborhood of the original point. If no CEX is found, the method returns "unknown". Hence, the method is still incomplete, but it aims at increasing the thoroughness of the analysis *w.r.t.* using only the approximate method based on abstract interpretation. Informally, it increases the chances of providing a definitive answer for each property.

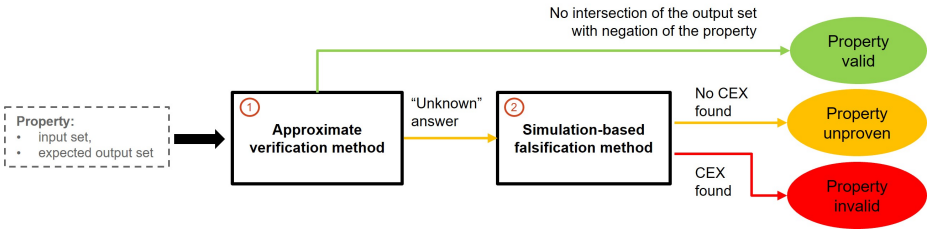

Fig. 3. Two-step verification method: abstract interpretation (first step) and simulation-based falsification (second step).

5 Verification Results

All experiments were run on a server with ∼64GB of RAM and a ∼2095Mhz Intel Xeon processor, in a Linux Ubuntu environment.

5.1 Reduction of Global Properties to Local Properties

Properties in Section 3 have been defined for *some* input time window $\mathbf{x} \in X$, X being the input space of the neural network (more precisely, its ODD). They can be imposed on the entire space X, making them global properties. Formal guarantees on the validity of these global properties would be desirable to demonstrate that the requirements are met. However, such analysis is currently intractable with existing VNN tools.

To mitigate the tool scalability problem, we propose an approximation of global property verification by reducing it to verifying *local properties* defined for a set of input points. The result of such reduction could be acceptable if altogether selected input points and corresponding local properties are representative of the RUL estimator ODD and cover it sufficiently. In other words, a discretization of the ODD could be performed with a representative set of input

points that approximate all possible input points within the ODD. Consequently, analysis of local properties for these points would approximate the verification of the corresponding global property.

According to data quality requirements prescribed by existing ML certification guidance for aviation applications [6], each dataset (i.e., training, validation, test) must be *complete* and *representative* with respect to the ODD. Statistical methods could be applied to assess these characteristics of the data [5]. Traditionally, testing of ML models is performed on a test (holdout) dataset. This dataset could be used as an approximation of the ODD, provided that its completeness and representativeness are demonstrated[3].

In this case study, we used the RUL estimator test dataset to perform the verification. Its quality has been analyzed and improved, as described in [5]. The test dataset includes 7493 time windows. They have been obtained by (i) concatenating degradation sequences (each degradation sequence is a multivariate time series that captures run-to-failure conditions of the bearing) in an arbitrary order and (ii) flattening this concatenation into a numbered list of time windows. That is, the inputs are temporally adjacent and consecutive, except that there is a discontinuity between the degradation sequences (last time window of the previous degradation sequence and first time window of the next sequence are not temporally adjacent). There is no specific ordering of degradation sequences in the test dataset, they are independent from each other.

5.2 Stability Verification

Several verification phases have been conducted for stability properties. First, the exact verification method was tested in the presence of a single CI perturbation at a single time step, as formulated in Equation 1. The encoding produced 52451 local stability properties. The framework was able to verify all properties with an average execution time of 0.36s; 100% of the properties were proven valid. Since all properties could be verified with a sound and complete method, it was considered redundant to test other verification methods in this phase.

Next, the exact method was employed to verify properties involving simultaneous perturbations of all CIs at a single time step, according to Equation 2. The encoding produced 7493 local properties, one per input time window. The method could not verify any property within a time limit of 3h, sometimes due to failures caused by out-of-memory events.

To mitigate the scalability problem, two-step verification method was applied to the same properties involving all CIs. The adoption of an approximate method allowed to overcome the difficulties experienced with the exact method: all properties were verified, 98.3% of them were proven valid, with an average verification time of 0.63s. Invalidity was detected, and an unknown answer was given, respectively, in 0.08% and 1.58% of the cases.

[3] A more detailed investigation of the reduction of global to local properties would include producing empirical evidence of the feasibility of the approach, as well as deriving analytical bounds on the approximation. This is subject of future work.

Results are summarized in Table 2, showing the total number of properties verified (or intended to be verified) at each verification phase, and a breakdown into amounts of valid, invalid, and unknown answers computed by the verification framework. Finally, average verification time per property is provided (avgTime) – cumulative, as well as average times for valid, invalid, and unknown properties (in brackets). Total verification time (totTime) is also provided in the last column. It can be noted that the use of the exact method to verify properties with perturbations in all CIs is redundant and in fact provided no results, however, it is also shown here to demonstrate the scalability issue. Since with the use of the two-step method invalid and unknown (unproven) properties are present, the table additionally specifies the results for the *critical range* of the RUL to see whether some of these properties could have potential impact on safety.

Table 2. Verification results for local stability properties of the RUL estimator.

Setting	**# prop**	**# valid**	**# inv**	**# unknown**	**avgTime (s)**	**totTime (s)**
Single CI, exact method	52451	52451	0	0	0.3255 (0.3255, -, -)	17074
All CIs, exact method	7493	-	-	-	Timeout (-, -, -)	Timeout
All CIs, 2-step method	7493	7369	6	118	0.6347 (0.5, 6.4, 4.1)	4756
All CIs, 2-step method **critical range**	1414	1414	0	0	0.3313	468.45

Fig. 4 shows where RUL estimator inputs that resulted in invalid and unknown stability properties are located within their degradation sequences. Recall that each property is associated with a time window, and that within each degradation sequence in the test dataset local stability properties correspond to temporally adjacent and consecutive time windows. Black solid vertical bars in Fig. 4 mark the ends of the degradation sequences: a vertical bar represents the end of a previous sequence, i.e., the failure of the bearing, and the beginning of a next one (full healthy state of the bearing). Red and blue dotted vertical bars identify, respectively, invalid and unknown properties.

It can be observed that invalid and unknown properties tend to be concentrated within the first half of the sequence time frame, i.e., the estimator behavior may be problematic at the very beginning of a degradation sequence. Noteworthy, there are no such properties in the critical range (last 100h of remaining lifetime of the bearing), where all properties are valid within the entire test dataset. This increases confidence in the correctness of the behavior of the RUL estimator with respect to safety considerations. Overall, it can be concluded that requirement RUL-ML-Stab-2 for stability to single CI perturbations is met, while RUL-ML-Stab-3 is not met.

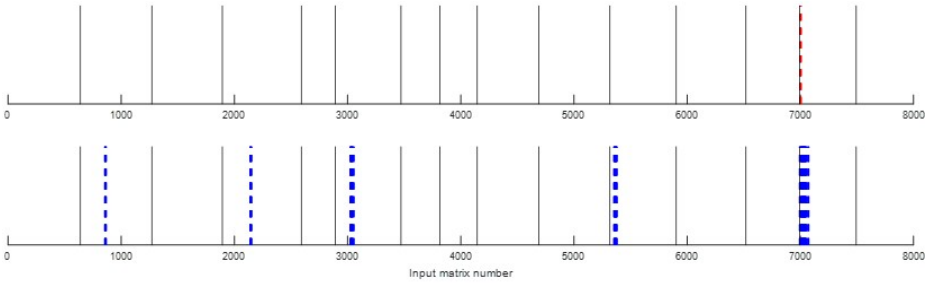

Fig. 4. Location of inputs related to invalid (top) and unknown (bottom) properties for all sequences in the test dataset.

Counterexample. It is important to analyze counterexamples to gain insights on the possible reasons of property violation. The verification framework permits visual analysis of the counterexamples. An example is illustrated in Fig. 5 for a stability property to multiple (all) CI perturbations at a single time step. It comes from the simulation-based falsification (second step of the two-step method), i.e., it is the result of applying bounded random perturbations on selected CIs corresponding to the property.

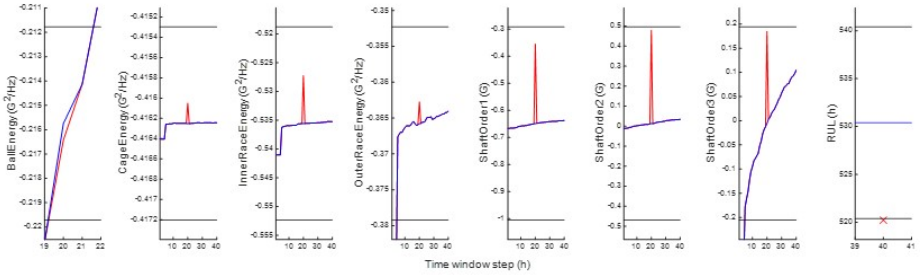

Fig. 5. Visualization of a counterexample to a local stability property.

First seven sub-charts in Fig. 5 show the CIs trends (in blue) over the input window. A perturbation to each of the CIs at step 20 is shown in red color. Black horizontal bars denote the perturbation bounds for each CI ($\pm\delta$). The last sub-chart on the right shows the original RUL estimator output (in blue) and the output deviation (red cross) due to input perturbations; the output exceeds the admissible deviation $\epsilon = 10$h. Several observations can be made:

- Each CI received a positive perturbation, i.e., a spike towards higher values that are associated with higher bearing degradation.
- The last three CIs (ShaftOrder 1-3) got perturbed more than others. The perturbation applied to these CIs is almost the maximum admissible one

(δ). The shaft is a cross-component for the bearing, so if the shaft health decreased substantially, as manifested by the ShaftOrder CIs, it would have a multiplicative effect on the degradation coefficient, which could significantly decrease the RUL. However, a spike increase in the shaft CIs at a single time step should not lead to such decrease as resulted from the CEX in Fig. 5.

5.3 Monotonicity Verification

Analysis has been performed on monotonicity properties with constrained fluctuation defined by Equations 3a-3b - both for each single CI slope change (and corresponding constrained fluctuation interval of CI trajectories) and for all CI slopes changed simultaneously, with γ varying from 10% to 50%. Again, the exact verification method did not provide any answer in a 3h timeout, therefore, the two-step method was applied.

Table 3 provides the results for growth rate change in a single CI. It reports numbers of valid, invalid and unknown properties, average verification times per property, and total verification times, for different CI growth trajectories (slopes) regulated by the γ parameter. Statistics for properties corresponding to input points in the critical range of the RUL (last 100h) are shown separately in the same table. Similarly, results for properties involving simultaneous growth rate change in all CIs are provided in Table 4. Following observations can be made:

- The estimator is more likely to correctly react, i.e., exhibit monotonic behavior, to simultaneous changes in all CIs rather than in an individual CI. Considering increased growth rates for individual CI and all CIs, the total percentage of valid properties in the former case varies between 79% and 84% (depending on γ), while for the latter 98%-99.5% of properties are valid. This is because multiple increasing CI trends in the input window provide more "evidence" to the estimator that the bearing is degrading.
- The estimator is more monotonic when the CI growth rate change is large: γ =50% has much fewer property violations compared to γ =10%. Larger changes make it more evident to the estimator that the degradation is happening. This holds for both changes in single CI and in all CIs.
- For single CI slope changes, property violations, as well as "unknown" answers, are uniformly distributed across the RUL range. For changes in all CIs, violations and "unknowns" mainly belong to the critical range of RUL.
- Average solving time for verification of a single property is a fraction of a second for individual CIs and around 1 second for all CIs. Verification of the latter is more complex since more inputs are considered variable (as intervals) rather than fixed: for a time window length of 40, individual CI properties result in 40 intervals to be defined (one per time step for a single CI), while 280 intervals (40×7 CIs) must be considered for all CI properties.

Same as for stability properties, monotonicity verification can generate counterexamples for invalid properties. Such CEX can be fed back to the designer for further analysis. An example is shown in Fig. 6 for one of the input windows

in the critical range of the RUL. Here, the growth rate of all seven CIs has been increased by $\gamma = 10\%$. Original CI trajectories are shown in blue and modified ones in red. The RUL value (red cross on the rightmost plot) is larger than the original one (blue horizontal line), therefore, the property is violated. In fact, even though the new CI trajectory with higher CI growth rate has values strictly larger than the original one, the estimator fails to predict a decreasing RUL. One can observe non-monotonicity in some of the CIs (e.g., BallEnergy and CageEnergy; they are present even in the original trajectories). These fluctuations may be due to flight regime or mission change within the time window.

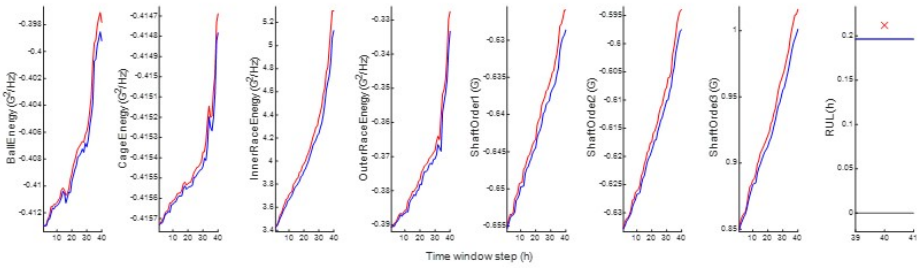

Fig. 6. Counterexample for a monotonicity property with a 10% growth rate increase for all CIs.

Table 3. Monotonicity property verification results with two-step method (single CI, increased CI growth rates).

range	γ	**# prop**	**# valid**	**# invalid**	**# unknown**	**avgTime (s)**	**totTime (s)**
full	10%	52451	41572	7797	3082	0.1716	8999
	20%		42875	7624	1952	0.1660	8706
	30%		43513	7568	1370	0.1591	8346
	40%		43869	7543	1039	0.1632	8563
	50%		44108	7524	819	0.1608	8436
critical	10%	9898	8061	1455	382	0.1620	1604
	20%		8248	1446	204	0.1568	1552
	30%		8316	1445	137	0.1498	1483
	40%		8359	1434	105	0.1533	1517
	50%		8387	1426	85	0.1515	1500

Overall, verification of local monotonicity properties defined from requirements RUL-ML-Mon-1 and RUL-ML-Mon-2 reveals that the current version of the ML model does not meet neither of the two requirements. This is due to:

– A significant number of property violations occurs on relatively small changes of the CI growth rate (e.g., γ =10%) – 20% for all single-CI properties, with

Table 4. Monotonicity property verification results with two-step method (all CIs, increased CI growth rates).

range	γ	**# prop**	**# valid**	**# invalid**	**# unknown**	**avgTime (s)**	**totTime (s)**
full	10%	7493	7364	59	70	1.049	7858
	20%		7458	26	9	1.075	8052
	30%		7457	25	11	1.068	8005
	40%		7458	24	11	1.104	8274
	50%		7458	22	13	1.080	8098
critical	10%	1414	1373	26	15	0.984	1392
	20%		1379	26	9	1.010	1429
	30%		1378	25	11	1.004	1419
	40%		1379	24	11	1.027	1452
	50%		1379	22	13	1.008	1426

the majority of invalid properties belonging to the changes in ShaftOrder1 and ShaftOrder3. Additionally, there is a large number of unknown properties, also in the critical RUL range. This shows that the estimator is often not capable of associating growing CI trends with component degradation.

– Despite the number of invalid properties for growth rate changes in all CIs (RUL-ML-Mon-2) being small (around 0.3%), all violations belong to the critical region of the RUL.

Limited non-monotonicity. Additional analysis has been carried out to understand to which extent monotonicity properties are not satisfied, e.g., if such increase of the RUL value can be bounded from above. For this purpose, the notion of *limited non-monotonicity* has been introduced and encoded. For properties with single CI growth rate increase (Eq. 3) this is formalized as follows:

$$\forall \boldsymbol{x'} : \forall t : u_i^t \leq x_i^{'t} \leq max(u_i^t, u_i^{t+1}) \rightarrow f(\boldsymbol{x'}) \leq f(\boldsymbol{x}) + \epsilon \tag{4}$$

for step t, constant γ, constant $\epsilon = 10$h (provided by the domain expert), $u_i^{t+1} = x_i^{t+1} + \gamma|x_i^{t+1} - x_i^1|$, S being the subset of indexes i of features corresponding to CIs. Equation 4 imposes that a constrained fluctuation in the CIs yields an increase of the estimated RUL that is limited by a constant number of hours ϵ.

Analysis of the relaxed property allowing limited non-monotonicity has been executed only for properties reported as invalid or unknown during verification activities presented above. Numerical results are available in [5], pages 87-88. Results for single CI have shown that the majority of the previously invalid/unknown properties become valid, i.e., the violations of the original properties are mostly bounded by $\epsilon = 10$h. A small number of properties remained invalid. The analysis showed that they belong to a group of adjacent time windows in a single degradation sequence. Violations may be related to fluctuating behavior of the CI within the corresponding windows, possibly due to errors in simulations that produced synthetic data for the test dataset.

6 Conclusion and Future Work

We applied formal methods on an industrial case study to verify properties of a deep learning based estimator of remaining useful life. We provided mathematical formalizations of stability and monotonicity properties of neural networks. To overcome scalability limitations of VNN tools, we proposed to reduce the verification of global properties to a representative set of local properties, and also implemented a two-step verification method involving abstract interpretation and simulation-based falsification. Numerical results demonstrate the applicability of formal methods to verify a large number of local properties in reasonable time. Future work shall focus on further improving both the completeness of the analysis and its scalability, derivation of an error bound for the reduction of global properties with local properties, as well as on the application of FM on higher complexity neural networks, such as perception systems.

Acknowledgements The authors wish to thank Eric DeWind and David F. Larsen for fruitful discussions and feedback about the RUL estimator, as well as for providing mechanical bearing degradation datasets to train and test the neural network.

References

1. Remaining Useful Life Estimation using Convolutional Neural Network. [Online]. https://www.mathworks.com/help/releases/R2021a/predmaint/ug/remaining-useful-life-estimation-using-convolutional-neural-network.html
2. Similarity-Based Remaining Useful Life Estimation. [Online]. https://www.mathworks.com/help/predmaint/ug/similarity-based-remaining-useful-life-estimation.html
3. Benkedjouh, T., Medjaher, K., Zerhouni, N., Rechak, S.: Remaining useful life estimation based on nonlinear feature reduction and support vector regression. Engineering Applications of Artificial Intelligence **26**, 1751–1760 (2013)
4. 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: Computer Safety, Reliability, and Security: 40th International Conference, SAFECOMP 2021, York, UK, September 8–10, 2021, Proceedings 40. pp. 34–48. Springer (2021)
5. EASA and Collins Aerospace: Formal Methods use for Learning Assurance (ForMuLA). Tech. rep. (April 2023)
6. European Union Aviation Safety Agency (EASA): Concept Paper: Guidance for Level 1&2 Machine Learning Applications. Concept paper for consultation. (February 2023)
7. Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., et al.: The Marabou framework for verification and analysis of deep neural networks. In: Computer Aided Verification: 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I 31. pp. 443–452. Springer (2019)

8. Li, X., Ding, Q., Sun, J.Q.: Remaining useful life estimation in prognostics using deep convolution neural networks. Reliability Engineering & System Safety pp. 1–11 (2018)
9. Liu, C., Cofer, D., Osipychev, D.: Verifying an Aircraft Collision Avoidance Neural Network with Marabou. In: Proceeding of NASA Formal Methods Symposium (2023)
10. Pecht, M., Gu, J.: Physics-of-failure-based prognostics for electronic products. IEEE Transactions of Measurement and Control **31**, 309–322 (2009)
11. Ren, L., Cui, J., Sun, Y., Cheng, X.: Multi-bearing remaining useful life collaborative prediction: A deep learning approach. Journal of Manufacturing Systems **43**, 248–256 (2017)
12. RTCA/DO-178C: Software Considerations in Airborne Systems and Equipment Certification (2011)
13. RTCA/DO-333: Formal Methods Supplement to DO-178C and DO-278A (2011)
14. SAE G-34 Artificial Intelligence in Aviation: Artificial Intelligence in Aeronautical Systems: Statement of Concerns (2021)
15. Tran, H.D., Yang, X., Manzanas Lopez, D., Musau, P., Nguyen, L.V., Xiang, W., Bak, S., Johnson, T.T.: NNV: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In: Computer Aided Verification: 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21–24, 2020, Proceedings, Part I. pp. 3–17. Springer (2020)
16. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Efficient formal safety analysis of neural networks. Advances in neural information processing systems **31** (2018)