Innovation Network
EASA AI Task Force

Collins Aerospace

Report

Formal Methods use for Learning Assurance (ForMuLA)

Public extract

April 17, 2023

Version 1.0

Authors (in alphabetical order)

Collins Aerospace

Dr. Massimo Baleani
Arthur Clavière
Dr. Darren Cofer
Eric DeWind
Dr. Luigi Di Guglielmo
Dr. Orlando Ferrante
Giacomo Gentile
Dr. Dmitrii Kirov*
David F. Larsen
Dr. Leonardo Mangeruca
Dr. Simone Fulvio Rollini*

ipc-formula@collins.com

EASA

Giovanni Cima
Ralf Schneider
Hassan Semde*
Guillaume Soudain

ai@easa.europa.eu

Citing this report

EASA and Collins Aerospace, *Formal Methods use for Learning Assurance (ForMuLA)*, April 2023.

```
@techreport{ForMuLA,
       author = {EASA and Collins Aerospace},
       title = {Formal Methods use for Learning Assurance (ForMuLA)},
       month = 4,
       year = 2023
}
```

Disclaimer

This document and all information contained or referred to herein are provided for information purposes only, in the context of, and subject to all terms, conditions and limitations expressed in the IPC contract P-EASA.IPC.017 of September 20th, 2021, under which the work and discussions to which they relate were conducted. Information or opinions expressed or referred to herein shall not constitute any binding advice, nor they shall create or be understood as creating any expectations with respect to any future certification or approval whatsoever.

* Principal authors

Table of Contents

Acronyms

ASAM	Association for Standardization of Automation and Measuring Systems
ATG	Automated Test Generation
CAMO	Continuing Airworthiness Management Organization
CBM	Condition-Based Maintenance
CEX	CounterEXample
CI	Condition Indicator
CNN	Convolutional Neural Network
ConOps	Concept of Operations
DAL	Design Assurance Level
DL	Deep Learning
EASA	European Union Aviation Safety Agency
EUROCAE	EURopean Organization for Civil Aviation Equipment
FAA	Federal Aviation Administration
ForMuLA	Formal Methods use for Learning Assurance
FM	Formal Methods
HUMS	Health and Usage Monitoring System
HW	Hardware
ILP	Integer Linear Programming
IPC	Innovation Partnership Contract
MBSE	Model-Based Systems Engineering
MILP	Mixed Integer Linear Programming
ML	Machine Learning
MoC	Means of Compliance
MRO	Maintenance, Repair and Overhaul
MTS	Multivariate Time Series
NN	Neural Network
ODD	Operational Design Domain
OMT	Optimization Modulo Theories
ONNX	Open Neural Network Exchange
PDF	Probability Density Function
PHM	Prognostics and Health Management
ReLU	Rectified Linear Unit
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
RTA	Run-Time Assurance
RUL	Remaining Useful Life
SAE	Society of Automotive Engineers (company; current name – SAE International)
SME	Subject Matter Expert
SMT	Satisfiability Modulo Theories
SVM	Support Vector Machine
SW	Software
VHS	Vehicle Health System
VNN	Verification of Neural Networks
V&V	Validation and Verification

Glossary

Formal methods. Formal methods are mathematically based techniques for the specification, development, and verification of software aspects of digital systems. The mathematical basis of formal methods consists of formal logic, discrete mathematics, and computer-readable languages. The use of formal methods is motivated by the expectation that, as in other engineering disciplines, performing appropriate mathematical analyses can contribute to establishing the correctness and robustness of a design [1].

Learning assurance. All of those planned and systematic actions used to substantiate, at an adequate level of confidence, that errors in a data-driven learning process have been identified and corrected such that the system satisfies the applicable requirements at a specified level of performance, and provides sufficient generalisation and robustness guarantees [2].

Operational Design Domain (ODD). Operating conditions under which a given AI-based system is specifically designed to function as intended, in line with the defined ConOps, including but not limited to environmental, geographical, and/or time-of-day restrictions. ODD defines the range of operating parameters within which the AI-based system is designed to operate, and as such, will only operate nominally when the parameters described within the ODD are satisfied. The ODD also considers correlations between operating parameters in order to refine the ranges between these parameters when appropriate; in other words, the range(s) for one or several operating parameters could depend on the value or range of another parameter [2].

Data completeness. A dataset is *complete* if it sufficiently (i.e., as specified in the data quality requirements) covers the entire space of the operational design domain for the intended application [2].

Data representativeness. A dataset is *representative* when the distribution of its key characteristics is similar to the actual input state space for the intended application [2].

ML model[1]. Mathematical model that is generated as an output of a learning algorithm. Its parameters are determined during the training process and fixed after it is finished.

ML inference model. Implementation of the trained ML model on the target platform (software and/or hardware).

ML constituent. A collection of traditional hardware and/or software items (e.g., pre-processing and post-processing elements) and at least one specialized hardware or software item that contains one or more ML inference models.

ML constituent ODD. Operating conditions under which a given ML constituent is expected to work as intended.

Learning algorithm stability. Learning (training) algorithm is *stable* if in the presence of perturbations in the dataset *in the training phase* of learning assurance (e.g., replacement/removal of data points, additive noise, labelling errors) it produces a model that is similar, in terms of its properties and characteristics, to the one trained on the original dataset. *For example, for classification models this would mean that model decision boundaries do not change significantly in case of training dataset perturbation.*

[1] Sometimes in this report it is also referred to as "*trained* ML model".

ML model stability. ML model is *stable* if a small, bounded perturbation applied to its inputs *in normal operating conditions*, i.e., when the inputs are inside the ML constituent's operational design domain, does not cause a significant deviation in its output[2].

ML model robustness. ML model is *robust* if it does not exhibit unexpected behavior neither in normal operating conditions (i.e., the model is stable) nor in adverse conditions, for example, in response to out-of-distribution inputs, adversarial inputs, or edge/corner cases.

ML model generalization. The capability of the ML model to exhibit required performance on *unseen* inputs within its operational design domain, i.e., those inputs that have not been part of training and validation datasets. *Generalization is typically evaluated using a test (holdout) dataset in order to demonstrate that the model has a reasonable bias-variance trade-off, i.e., does not underfit or overfit the training data. While such testing approach measures ML model generalization capability with respect to the chosen test dataset, additional analytical approaches may be required to assess model generalization to the entire admissible input space (e.g., see the discussion in* [3]*).*

[2] In general, a discontinuity in the ML model's output does not necessarily represent its instability. If the phenomenon/process that the ML model describes has known discontinuities, i.e., large changes in the output in some input region, these should be documented and considered expected during the ML model stability assessment.

List of figures

List of tables

Executive summary

The aim of this report is to present the outcome of the collaboration between EASA and Collins Aerospace on an Innovation Partnership Contract (IPC) that investigated the use of formal methods as part of the learning assurance building block of the EASA AI Roadmap [4]. The project ran from Oct 2021 to Mar 2023.

The IPC project titled **"Formal Methods use for Learning Assurance" (ForMuLA)** focused on emphasizing opportunities for the adoption of formal methods techniques in the design assurance process of machine learning enabled systems. This resulted in the following key achievements:

- Proposed use of formal methods as anticipated means of compliance for a set of key certification objectives from the EASA Concept Paper for Level 1&2 Machine Learning Applications. This supported the update of definitions in the concept paper and the clarification of objective LM-11 on learning algorithm and trained model stability, which has been split into objectives LM-11 and LM-12 in the transition to the new version of the concept paper.
- Detailed discussion of relevant formal methods (FM) technologies and supporting statistical methods, and their possible role in the development and validation and verification (V&V) of machine learning enabled systems. Emphasis has been made on innovative FM applications specific to the robustness assessment of machine learning models.
- Practical demonstration of the use of formal methods on an industrial use case of a deep learning-based estimator for remaining useful life of mechanical bearings in airborne equipment. The output of the estimator is used for on-ground maintenance applications. Demonstrations provided concrete evidence of how FM and supporting statistical techniques can be used as part of the verification activities to deal with data quality assessment, ML stability, robustness and intended behavior verification.

The considerations summarized in the report apply to machine learning in general, but particular emphasis has been placed on specific challenges related to neural networks. Discussion of formal methods applications are purposefully kept generic. This report does not recommend specific methods or tools, but is rather intended to motivate opportunities from a theoretical perspective. Where applicable, a reference is made to concrete methods and tools.

European Union Aviation Safety Agency (EASA) is the centerpiece of the European Union's strategy for aviation safety. Its mission is to promote the highest common standards of safety and environmental protection in civil aviation. The Agency develops common safety and environmental rules at the European level. It monitors the implementation of standards through inspections in the Member States and provides the necessary technical expertise, training and research. The Agency works hand in hand with the national authorities which continue to carry out many operational tasks, such as certification of individual aircraft or licensing of pilots.

Collins Aerospace, a Raytheon Technologies company, is a leader in technologically advanced and intelligent solutions for the global aerospace and defence industry. Collins Aerospace has the capabilities, comprehensive portfolio, and expertise to solve customers' toughest challenges and to meet the demands of a rapidly evolving global market. The **Applied Research & Technology (ART)** organization of Collins Aerospace is an agile centrally held enterprise level technology organization that works to identify, develop and demonstrate innovative technology solutions, products, services, and intelligent systems supporting Collins Aerospace businesses with the vision of "accelerating transformative technologies for a safer more connected and sustainable world". As part of the ART organization, the Advanced Model Based Engineering Methods (AM2) department works to develop, mature, and transfer model-based methods, technologies and tools from conception to validation and verification of Collins products from sales to operation.

1 Introduction

Artificial Intelligence (AI) in aviation is a disruptive technology that will impact various products and services. 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. This is achieved by providing design assurance, i.e., evidence that certain guidelines and verification processes have been followed during the design process or that the product possesses necessary safety features (e.g. redundancy, runtime monitors, or safety nets).

The aviation industry currently does not have a consensus on the design assurance of ML constituents because they are not fully amenable to current design assurance processes and standards. In particular, ED-12C/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* [5]. 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 shall be 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 constituent development, instead, is *data-driven*. An ML model is trained through a learning procedure that starts from data, not from requirements[3]. Thus, the use of traceability of the implementation back to requirements as a means to minimize the risk that the ML constituent includes unintended behaviors is not effective. Additionally, the use of structural coverage metrics may not be effective in identifying unintended behavior in ML models such as neural networks (NN) [6]. Instead, as part of learning process verification activities, ML generalization and robustness assessment have been proposed as key criteria that, when fulfilled, can help to mitigate such risks in these typically “black-box” systems [2]. Therefore, it is critical to identify promising methods to evaluate generalization and robustness of ML models.

This report provides a theoretical overview and practical demonstrations of how formal methods (FM) techniques can be leveraged in the design assurance process of ML-enabled systems, also called learning assurance [2], with particular emphasis on the learning process verification activities dealing with ML stability and robustness.

According to ED-216/DO-333 [1], formal methods 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. It is worth noting that formal methods provide *comprehensive assurance* of properties for those aspects that are formalized in the formal model. The key requirement of any formal verification method is *soundness*: only properties that are actually valid shall be declared valid by a sound method (see the full definition in Section 3.2).

The report summarizes the progresses that the research community is achieving in identifying proper languages and formal models to capture ML models/constituents and their properties, as well as in developing novel formal verification algorithms and tools able to extend the applicability of traditional formal verification capabilities to the assurance of ML robustness and beyond.

1.1 Background

The path towards ML certification is not yet defined, but several reports have been published by aviation authorities and research groups addressing foundational certification aspects of ML-enabled systems. In February 2020 the European Aviation Safety Agency (EASA) published their *Artificial Intelligence Roadmap*,

[3] In fact, datasets may *implicitly* represent some functional requirements for the system.

including the timeline shown in Figure 1. The roadmap establishes the Agency's initial vision for the safety and ethical dimensions of development of ML in the aviation domain [4]. Its main scope is to create a framework for ML trustworthiness and establish conditions for use of ML applications in aviation.

In March 2020 and May 2021 EASA released two technical reports entitled *Concepts of Design Assurance for Neural Networks* (CoDANN) I and II [3] [7]. These reports provide a detailed study of several ML-specific development and assurance aspects such as robustness, generalization and explainability topics, and propose a W-shaped ML development process (***Figure 2***) outlining the essential steps for learning assurance.

In March 2021, the DEEL (Dependable and Explainable Learning) certification working group published a whitepaper entitled *Machine Learning in Certified Systems* [8] summarizing the necessary conditions and objectives for certifying ML-based systems. Auditability, data quality, explainability, maintainability, robustness, resilience, specifiability and verifiability are the identified certification objectives.

In April 2021, the SAE G-34/EUROCAE WG-114 joint international committee on Artificial Intelligence in Aviation published a statement of concern document reviewing "*current aerospace software, hardware and system development standards used in the certification/approval process of safety-critical airborne and ground-based systems,"* and assessing *"whether these standards are compatible with a typical artificial intelligence (AI) and machine learning (ML) development approach"* [6]. This was followed by a technical paper [9] presenting a new ML development lifecycle which will constitute the core of the new aeronautical standard on ML called AS6983 jointly being developed by EUROCAE and SAE. The paper covers the design assurance process at the item level (analogously to ED-12C/DO-178C for traditional avionics software) and proposes development and V&V lifecycle activities compatible with the ones identified by EASA.

In December 2021, EASA published a concept paper titled "First Usable Guidance for Level 1 Machine Learning applications in aviation," [10] which is the first milestone in the implementation of the EASA Roadmap. The guidance anticipates a set of assurance objectives (compatible with the ones proposed by DEEL), and additionally proposes means of compliance supporting applicants in the identification of certification means for ML-based safety-critical systems. In February 2023 EASA has published the update of the concept paper, extending it to Level 2 ML applications [2].

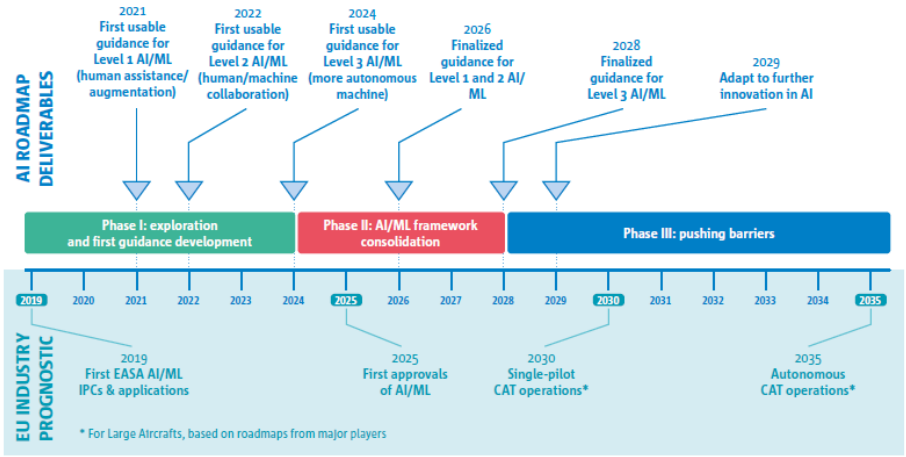

Figure 1. *EASA AI Roadmap.*

1.1.1 IPC: CoDANN I and CoDANN II

One important aspect of the development of machine learning systems is that it is based on a data-driven process. Specific activities on data management (data collection, data preparation, and data quality verification) need to be present in the process. Furthermore, the development process cannot go directly from requirements and data to programming, but must include a new paradigm of *learning*, i.e., creating and

training a mathematical model (ML model) from the data. This model can then be transformed to software and deployed on a target computer. As part of the two IPCs, titled Concepts of Design Assurance for Neural Networks (CoDANN I and II) [3] [7], a new AI development process, called the W-model, has been proposed. It is illustrated in ***Figure 2***.

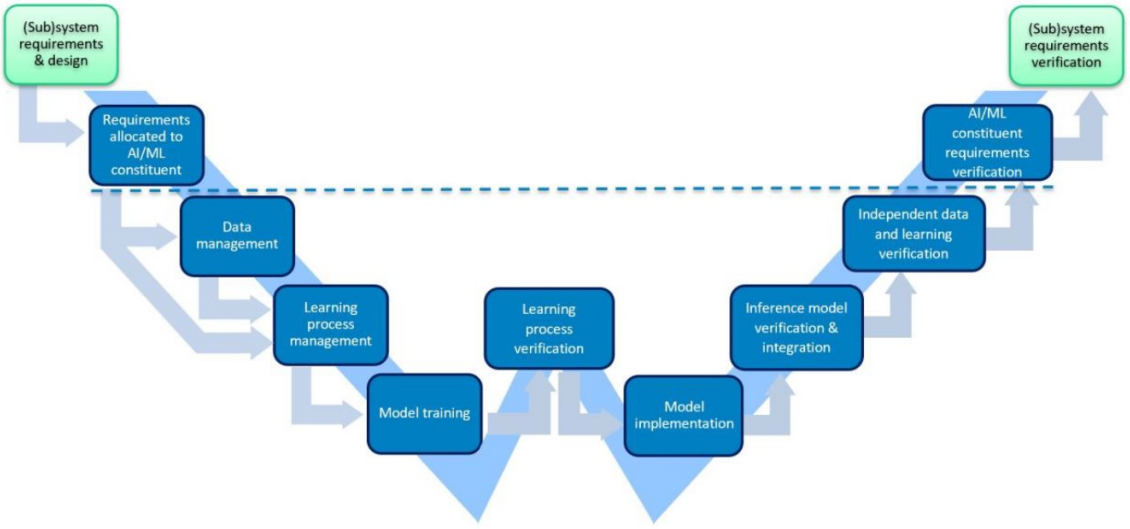

Figure 2. *W-shaped development cycle for machine learning (W-model).*

The left-hand side of the W-model covers the development activities specific to ML components, including the following:

- Data management, which includes collection of the data (e.g., real, or synthetic data), its pre-processing (e.g., normalization, filtering, feature selection, annotation, labelling), and data quality verification.
- Learning process management, which covers all steps required prior to training the model, such as model architecture selection, training algorithm, quality metrics, and hyperparameters.
- Model training, a self-explanatory step in ML to find a best-performing model.

The key element in the process, which transforms the V into a W, is the Learning process verification, where specific tests and analyses must be applied to ensure that the trained model meets the key criteria – generalization and robustness. The former means that the model performs well on previously unseen data. The latter demonstrates that the performance of the model does not degrade in case of perturbations applied to its inputs and in case of adverse inputs, such as adversarial attacks and out-of-distribution data. Both criteria are key challenges in ML and must be fulfilled to demonstrate the absence of unintended functionality in these typically black-box systems, which is a crucial condition for their certification by aviation authorities.

The model verification step is followed by the second “V” of the W-model, with one more development activity, Model implementation. This phase covers creation of code that implements the trained model (also called the *inference model*) and its deployment on target hardware, as well as verification and integration activities of the inference model to ensure that the properties of the trained model are preserved in the deployed version.

1.1.2 EASA concept paper for level 1&2 ML applications

The EASA's Concept Paper: Guidance for Level 1&2 ML Applications[4] "*introduces a first set of objectives, in order to anticipate future EASA guidance and/or requirements to be complied with by safety-related ML applications. Where practicable, a first set of anticipated Means Of Compliance (MOC) has also been developed, in order to illustrate the nature and expectations behind the objectives. The aim is to provide applicants with a first framework to orient choices in the development strategy for ML solutions*" [2]. Among the proposed MOCs, formal methods are considered promising, especially for the verification of stability and robustness properties of the trained and inference models.

The concept paper also provides the following definition of learning assurance: "*All of those planned and systematic actions used to substantiate, at an adequate level of confidence, that errors in a data-driven learning process have been identified and corrected such that the system satisfies the applicable requirements at a specified level of performance, and provides sufficient generalisation and robustness guarantees.*"

1.1.3 ED-216/DO-333 formal methods supplement

In commercial aviation, the use of formal methods in the development of software-based aerospace systems is allowed through the ED-216/DO-333 [1] supplement to the ED-12C/DO-178C [5] standard. The supplement identifies the modifications and additions to ED-12C/DO-178C objectives, activities, and software life cycle data that should be addressed when formal methods are used as part of the software development process. The standard highlights that the extent to which formal analysis can be used varies according to the ability to construct appropriate formal models, the choice of analysis techniques, and the availability of tools. Therefore, it is critical to define formal models provided with the appropriate level of detail representing a conservative approximation of the important software properties and of the properties stated by the natural language requirements, and to adopt tools able to process such models and perform analyses leveraging mathematical reasoning.

1.1.4 Limitations of formal methods

As discussed in DO-333, *applicability of formal methods to any particular software development activity is bounded by the ability to construct an appropriate formal model. Requirements specified in natural language may include properties that cannot be verified with a formal method. Models can also be insufficiently detailed to allow meaningful analysis of some properties and yet be perfectly adequate for others. Despite these limitations, formal methods can be a means of completely and accurately describing important software properties. Formal analyses can then be applied to provide assurance of these properties* [1].

In recent years, new approaches – including the use of formal methods – have been proposed for the verification of ML components and ML-enabled systems. Furthermore, new formal methods tools are in development that permit mathematical analysis of ML models, such as neural networks. These are currently limited by scale and the need to precisely define requirements for analysis. Current state-of-the-art tools today could not be applied to large open input spaces and complex ML models, such as vision-based models. However, they are making rapid progress and have been used and applied at scale for low-complexity models (e.g., shallow NNs) with well-defined input spaces to prove critical robustness properties for real systems. At the same time, the research community is actively working on overcoming the existing barriers of FM tools scalability by developing novel techniques and tools that may be able to address higher complexity ML verification problems in the future.

[4] For short, in the remainder of the report it is referred to as "EASA AI Concept Paper".

1.2 Scope of the ForMuLA project

The ForMuLA project focuses on identifying opportunities for adopting formal methods as means of compliance for the assurance and certification objectives for ML constituents. The following topics have been addressed:

- Identification of available state-of-the-art FM technologies specific to ML. Such technologies have been adapted to or specifically developed for machine learning and are able to support several development and V&V activities, including data quality assessment, improvement of the learning process, ML model stability and robustness verification,and explainability. For those activities, where FM may face scalability or applicability limitations, statistical methods have also been explored as supporting means for the analysis.
- Identification of key FM applications for the learning process verification, especially for the assessment of ML model stability, robustness, and verification of intended behaviors. To overcome current limitations of FM tools in verifying global properties, the project identifies a method that reduces the problem of global properties verification to a set of local pointwise verifications performed relying on a conservative approximation. This method could be acceptable if the selected points and corresponding properties are representative of the ML constituent ODD.
- Practical demonstration of the use of formal methods on an industrial use case of a deep learning-based estimator for remaining useful life (RUL) of mechanical bearings in aircraft equipment in support of on-ground maintenance activities. A number of innovative FM technologies have been selected based on their applicability to the use case and used to evaluate the ML model stability, robustness and compliance with intended behaviors. At the same time, selected technologies have been evaluated in terms of efficiency and effectiveness. Furthermore, the use of statistical methods for data quality assessment has also been demonstrated.

Based on both theoretical and experimental studies, the project has come up with proposals for the use of formal methods as anticipated means of compliance for a set of key certification objectives from EASA AI Concept Paper [2]. It supported the update of definitions in the concept paper and the clarification of objective LM-11 on learning algorithm and trained model stability, which has been split into objectives LM-11 and LM-12 in the transition to the new version of the concept paper.

1.3 Outline of the report

After the brief introduction provided in Section 1 summarizing the scope of the ForMuLA project and the related background, Section 2 presents a concrete use case that will be used to contextualize the discussions in the following sections of the document. The selected use case consists of an ML-based Remaining Useful Life (RUL) estimator which is used in the context of a Prognostics and Health Management (PHM) system. After a short description of what RUL is and what solutions can be adopted to implement it, the ConOps and the architecture of the ML-based system incorporating the RUL functionality are described. Several details are offered to clarify the ML constituent operational design domain, the system-level and the ML constituent requirements, the datasets, the ML model characteristics, and, finally, the safety considerations related to the use case.

Section 3 provides the necessary background on formal methods, covering their capabilities and range of applications. Definitions are offered to help the reader to understand the FM taxonomy and associated technologies. Examples complement the technology overview to guide the reader in gaining confidence on how formal methods can be used for validation and verification activities, as well as for supporting some development activities for ML. Complementary to FM technologies, selected statistical methods are discussed that can support the analysis for key ML assurance objectives for which FM are either not applicable or not scalable.

Section 4 is the core of the report, as it describes the innovative applications of formal methods in the context of ML assurance. These approaches go beyond the traditional V&V applications of FM that are described in existing standards (e.g., ED-216/DO-333). The intent is to stress that FM can be used as anticipated means of compliance for the objectives that address new challenges specific to ML, such as robustness of ML models, as prescribed by existing guidance.

Section 5 demonstrates the application of formal methods on the use case described in Section 2. The demonstration focuses on a selected subset of FM approaches that includes data representativeness assessment and verification of ML model stability, robustness, as well as several other use case specific properties defined based on the ML model requirements. An example toolchain is described to offer practical insight of how the existing technologies can be combined, and the experimental evaluations provide an idea of the applicability and the scalability of the described toolchain solution.

Finally, Section 6 (Conclusions) reviews the current accomplishments and discusses subjects for future work.

2 Concept of Operations and use case

2.1 ConOps and use case selection

2.1.1 Background

Remaining Useful Life (RUL) is a widely used metric in Prognostics and Health Management (PHM) that manifests the remaining lifetime of a component (e.g., mechanical bearing, hydraulic pump, engine). Existing RUL calculation procedures often use *physics-based degradation models* [11]. Such models typically include a set of coefficients, such as component degradation coefficient and operating environment coefficient, which can come from engineering knowledge or be estimated using system identification methods. Another group of methods are *similarity-based methods* [12], where condition indicators of the component are compared to degradation trends of similar components available from historical data.

While model-based approaches for RUL estimation tend to be more accurate if the complex system degradation is modeled precisely, they require extensive prior knowledge about physical systems, which is often unavailable in practice. Therefore, creation of accurate physics-based models may not be possible. Similarity-based methods often suffer from poor accuracy. In the recent decade, the focus has been gradually shifting towards *data-driven* approaches that are able to model the degradation characteristics based on historical sensor data and infer the underlying correlations and causalities in the collected data without relying on a physics model. Various machine learning approaches have been proposed [13] [14] [15], out of which the use of *deep learning* (DL) is of particular relevance [16]. The main advantage of applying DL in PHM is that highly nonlinear, complex multi-dimensional systems can be modeled without prior expertise on the system behavior provided that enough data is available. Raw sensor readings can be directly used as inputs to DL models, and their automatic feature extraction capabilities can be leveraged to discover the relationships between the inputs, the degree of impact on the RUL, as well as other contributions to the RUL that may be unknown to the expert [17]. This is similar, for example, to image data, where raw inputs to a neural network are pixels, while various features on the input image (e.g., presence of different objects, forms or shapes) are internally extracted by the network in the hidden layers. Therefore, DL methods require less domain expertise as they alleviate the need of feature engineering activities, which could be difficult and time-consuming, since it requires prior knowledge of machine health prognostics and signal processing.

The ML-based (more precisely, DL-based) RUL estimation component is expected to accept time series data – a sequence of input values taken at several subsequent time steps, i.e., within a time window [16]. Therefore, a 2D input is expected, where the first dimension corresponds to the number of time steps (e.g., historical) and the second dimension corresponds to the number of input features. Considering a reasonable time window size and all ML constituent ODD inputs (see ***Table 1*** and ***Table 2*** in Section 2.2.3), the total number of input values, i.e., the number of entries in the time window (number of features × number of time steps), may be on the order of 10^3, which is a high-dimensional input. **This motivates to apply a deep learning solution to this use case.** More details about the inputs are available in Section 2.2.

Additionally, current application scope **is limited to offline training only**. A trained ML model is assumed to be frozen, i.e., no further (online) updates to the model are made during operation.

Remark: *As mentioned above, DL can perform automatic feature extraction from raw data. Extracted features, such as, for example, filters in the hidden layers of a convolutional neural network, may not be interpretable by system developers and users. Understanding the meaning of such features shall play an important role both for developers and users of the ML system. Even though explainability is not the central topic of the ForMuLA report, its role in the development and for the end user are highly acknowledged, and relevant aspects are discussed in the report whenever applicable.*

2.1.2 ConOps alternatives

RUL estimation function can be used in the following application categories:

1. *On-ground*
    a. **Condition-Based Maintenance (CBM),** where the estimated RUL could contribute to such tasks as augmented manual inspection of components and scheduling of maintenance cycles for components, such as repair or replacement, thus moving from preventive maintenance to *predictive* maintenance (do maintenance only when needed, based on component's current condition and estimated future condition). This could allow to eliminate or to extend service operations and inspection periods, prevent unsafe component conditions, optimize component servicing (e.g., lubricant replacement), generate inspection and maintenance schedules, and obtain significant cost savings.
2. *In-flight*
    a. **Pilot decision support,** where estimated RUL is directly provided to the pilot via a dedicated screen/interface (e.g., cockpit display), so that they can be aware of the current state and remaining life of different components of the aircraft and take corresponding decisions in the scope of current flight mission. For example, low RUL value of an engine provides the pilot with critical information for managing a hazardous contingent situation, e.g., suggests to immediately abort the current mission (e.g., initiate emergency landing) or re-plan the mission (e.g., discard some objectives, land at the nearest runway, etc.).
    b. **Airborne software applications,** where RUL is communicated to other avionics software (SW) components, such as automated planners and decision makers. Such applications can use the RUL information to recommend appropriate mitigations (e.g., suggest an altitude change), direct the crew in case of missed recognition of hazardous situations, as well as provide real-time decision aids in the scope of the current flight mission.

Aforementioned applications of RUL estimation provide support to information analysis (RUL value is more comprehensible by the pilot/maintenance engineer than raw component sensor measurements or statistical condition indicators) and support to decision/action selection (pilot can use RUL to take mission-related decisions such as mission abort or re-planning; similar case for maintenance; decisions and mitigations can also be suggested by avionics software). Therefore, they map onto **Level 1A** and **Level 1B** in the EASA classification [2]. In future products (in particular, for 2b – Airborne applications) higher autonomy may be introduced, so that the use of RUL estimation function in Level 2 – Human-AI collaboration applications, may be expected.

It is currently premature to detail the use of in-flight RUL applications 2a-b, i.e., "real-time" RUL prediction of aircraft components, in the civil aviation context. First, rapid in-flight component degradation is unlikely due to periodic maintenance and inspections. Therefore, timely detection of degradation and possible failures is performed by on-ground operations. Second, real-time RUL monitoring appears to be more applicable in dynamic contexts with rapidly changing conditions (one may consider military applications or unmanned aerial vehicles with short dynamic missions). Finally, required Design Assurance Level (DAL) of avionic SW functions that predict RUL in-flight would likely be one of the highest, i.e., DAL A-B, which currently has many open challenges due to stricter assurance objectives. Therefore, ForMuLA IPC considers an on-ground application for RUL as a support for flight preparation.

2.1.3 Selected use case and associated ConOps

As discussed in Section 2.1.2, RUL estimation function is a PHM metric that shall be used for 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. Additionally, the failure probability of the component during the next flight mission could be estimated based on RUL.

RUL estimation discussed in the ForMuLA IPC is performed for a mechanical bearing component installed in the drivetrain assembly of a rotorcraft.

End users that are intended to interact with the ML-based RUL function include the **MRO** (Maintenance, Repair and Overhaul) team, the **CAMO** (Continuing Airworthiness Management Organization) team[5], and the **pilots**. MRO could use the RUL as a support for ongoing maintenance actions (e.g., collecting additional information on component's state and estimated remaining life), while CAMO could use it for planning/scheduling future inspection activities. The pilot could consult the estimated RUL of different aircraft components during pre-flight checks to detect possible problems and expected failures, so that they can be reported to on-ground services[6]. Pilots shall interact with the RUL function via a cockpit display – the function can be integrated into an existing display. MRO/CAMO users shall use a ground station display to consult the RUL value.

The CBM application of RUL estimation provides support to information analysis, because RUL value is more comprehensible by a human (pilot/MRO/CAMO) than raw component sensor measurements or statistical condition indicators. RUL can also provide support to decision/action selection: for example, during pre-flight check the pilot may decide to abort the departure and to communicate a possible component problem to MRO/CAMO; MRO may decide to prioritize some maintenance/inspection action; CAMO may decide to adjust the maintenance schedule of the aircraft. Therefore, the use case maps onto **Level 1A** and **Level 1B** in the EASA classification [2], because estimated RUL does not automatically drive the maintenance and inspection tasks, but only supports the human user in taking a related decision[7].

ML-based RUL estimator is a part of the Vehicle Health System (VHS) – a software system that monitors the health state of the aircraft and its components/subsystems. It constitutes a RUL estimation function to be implemented as an ML constituent (see ***Figure 3***). It accepts a set of statistical indicators (also called *condition indicators* – CIs) describing the state of the monitored component, as well as the information about the current flight phase, mission and environment, and outputs the predicted RUL value (time-to-failure). Therefore, ML model performs a *regression* task. Predicted RUL value corresponds to the remaining life of the monitored mechanical bearing and is provided to the pilot via a cockpit screen (see example in ***Figure 3a***), to the MRO/CAMO team member via a ground display, and to the failure prediction function that computes the current probability of failure of the component (the latter is out of scope of ForMuLA).

[5] CAMO is a civil aviation organization authorized to schedule and control continuing airworthiness activities on aircraft and their parts (https://en.wikipedia.org/wiki/Continuing_airworthiness_management_organization). A CAMO can also be the operator of the aircraft. The term CAMO is used in the European Union. CAMOs are audited by EASA. MRO performs the scheduled maintenance under the requirements of CAMO. Similarly, in the USA operators are required to have a *Continued Airworthiness Maintenance Program* (CAMP) that must be approved by FAA. MRO performs the scheduled maintenance under the requirements of the CAMP. Hereafter, the term CAMO is used for the entity that plans and schedules maintenance activities (for the US this would typically mean the operator of the aircraft).

[6] Note that ground services and pilots may require to observe the degradation trend within different timeframes (a longer trend may be required by MRO).

[7] In future products one may also expect higher autonomy levels, for example, for automated scheduling and optimization of maintenance cycles, automated pre-flight checks with single-pilot operations. Therefore, in the future, CBM application of RUL may also fall under **Level 2** – Human-AI collaboration. This is out of scope of the current use case and the ForMuLA project.

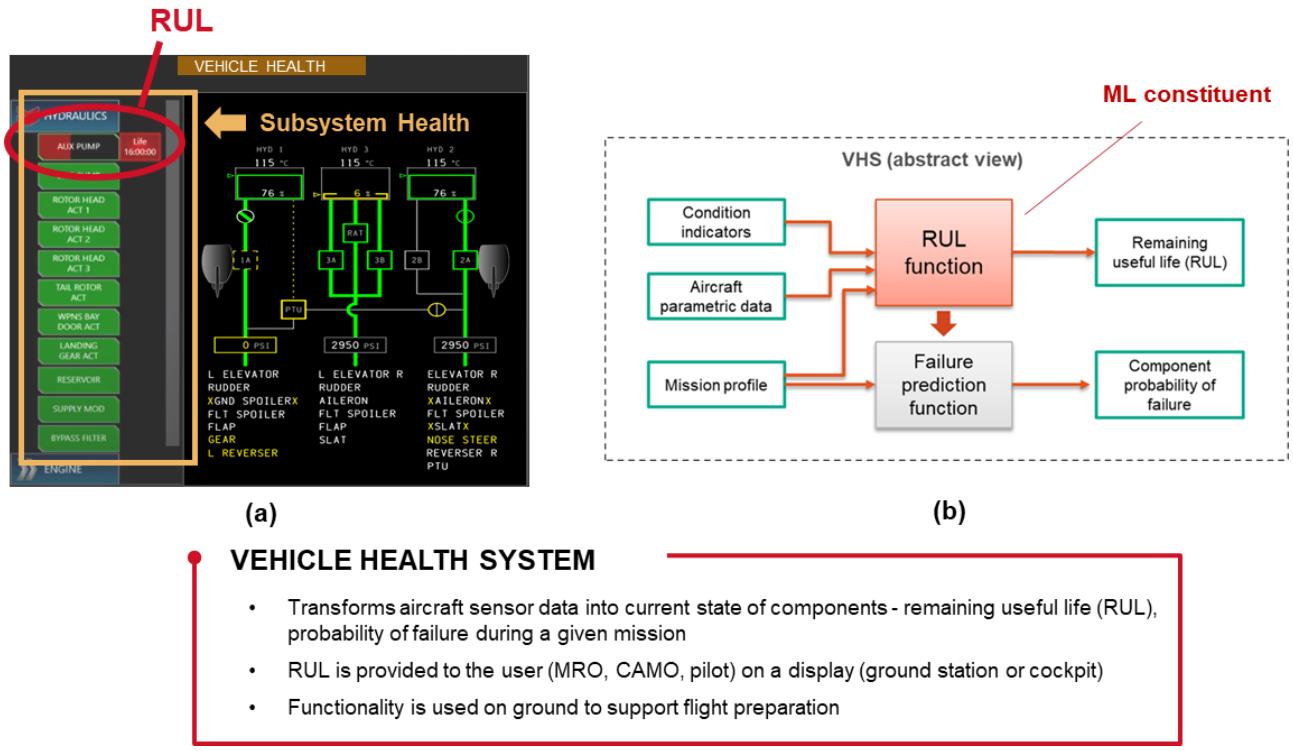

Figure 3. (a) VHS view on a cockpit display (example); (b) ML subsystem and ML constituent.

Remark: *VHS monitors the health state of different aircraft components (e.g., engines, bearings, hydraulic system, fans). These components may or may not have a dedicated RUL estimation function. If present, the RUL estimation function may be implemented both with and without ML. There may be multiple ML constituents implementing the RUL function, each of them dedicated to a different aircraft component. Each ML constituent is part of the ML subsystem that may also include functional elements implemented with traditional software, e.g., signal processing from raw sensor data to compute inputs for the ML constituent.*

2.2 Definition of the ML-based system

2.2.1 System architecture

VHS is an *ML-based system* since it includes a component/function based on machine learning, namely the RUL estimation function. It is assumed to be the only ML-based function in the VHS. The concept architecture of the VHS is illustrated in ***Figure 4***. It includes multiple subsystems/components, among which an *ML-based subsystem* is responsible for estimating the state of the mechanical bearing component mentioned above. It incorporates an ML constituent that is a deep learning based RUL estimator further detailed in Section 2.2.2. The constituent includes an ML-based RUL estimation function (ML inference model) and pre/post-processing elements implemented in traditional software. Health and Usage Monitoring System (HUMS)[8], as well as other avionics systems, provide inputs to the ML constituent. ML subsystem additionally includes traditional SW components that perform other, non ML-based functions, for example, estimation of failure probability of a mechanical component given its predicted RUL (out of scope of the ForMuLA project).

[8] HUMS is a generic term given to activities that utilize data collection and analysis techniques to help ensure availability, reliability and safety of vehicles (https://en.wikipedia.org/wiki/Health_and_usage_monitoring_systems).

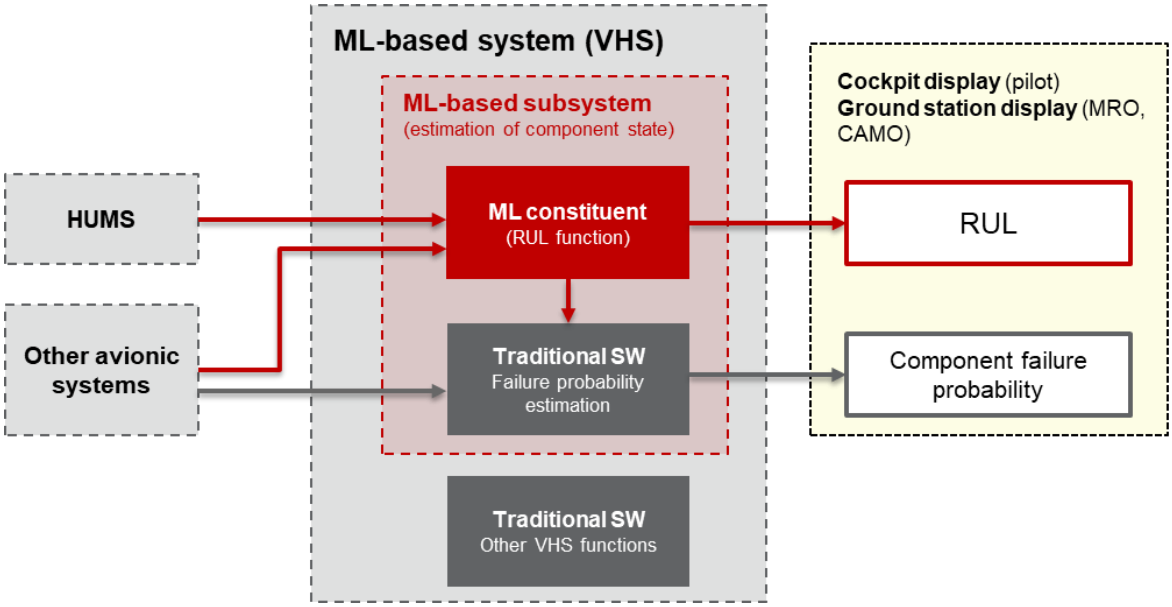

***Figure 4**. ML-based system, ML-based subsystem and ML constituent in the ForMuLA use case.*

2.2.2 Machine learning constituent

The use case of ForMuLA focuses on a single ML constituent that implements the RUL estimation function for a mechanical bearing installed in a drivetrain assembly of a rotorcraft. It is monitored by a vibration sensor, which is a single axis piezoelectric accelerometer mounted on the outside of the gearbox near the bearing. From this sensor measurements, a set of Condition Indicators (CIs) is computed by the HUMS using signal processing algorithms (this computation is done outside of ML constituent). A typical CI is an energy value, e.g., Ball Energy for the bearing, that manifests some degradation pattern.

Additional inputs are provided by other avionics systems: current flight regime (e.g., ascent, cruise), current mission, and current environment. The main factor that affects bearing degradation and its RUL is *how* the component is used. This depends on the load of the bearing, which is different across flight regimes, and, consequently, on the flight missions that the aircraft executes, because each mission is a sequence of flight regimes; also duration of each regime varies across missions. A set of mission patterns (types) that may be executed is known and specified in the operational design domain of the aircraft. RUL is also dependent on the environment conditions.

Inputs to the ML constituent represent a time window, i.e., an ordered sequence of snapshots of bearing state (represented by CIs and other quantities described above). Snapshots can be recorded both between and during flight missions. The data is recorded with a fixed time step and stored in memory to be later used in predictions[9].

Figure 5 illustrates the ML constituent that includes the ML model (deep learning; see Section 2.2.6 for description) and pre-/post-processing components implemented in traditional software and used for feature computation, (de-) normalization, monitoring of ML constituent ODD, and other relevant tasks. The output of the RUL estimator is provided to human users (MRO, CAMO, pilot) for CBM purposes, and to the component failure probability estimation function. Data recording and generation/update of time windows is performed outside of the ML constituent and, therefore, it is not part of its preprocessing functions.

[9] All inputs to the RUL function are also available during flight. This means that, in principle, RUL estimation can also be performed in-flight. Corresponding airborne applications are not part of the use case discussed in this report.

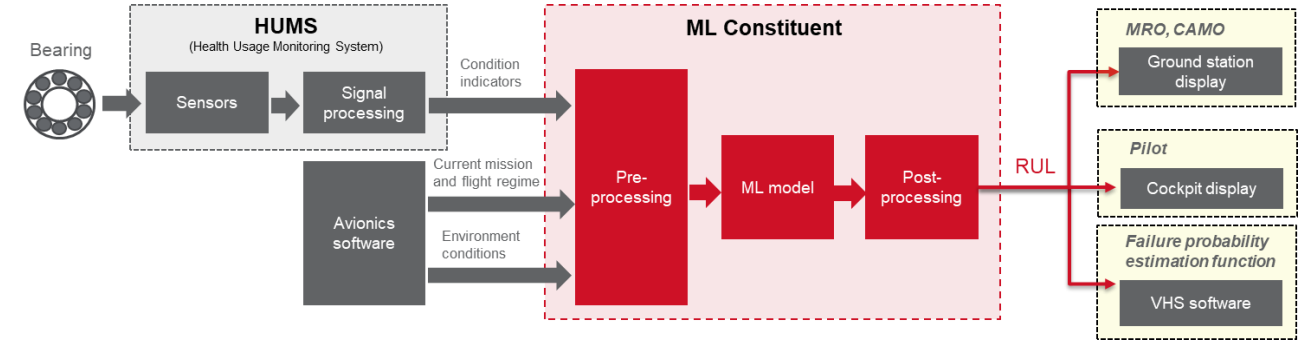

Figure 5. *Overview of the ML constituent for RUL estimation, and its operating environment.*

2.2.3 ML constituent operational design domain

2.2.3.1 Definition

Operational design domain of the ML constituent is defined at its boundary. Dimensions of the ML constituent ODD, as well as their expected limits and/or probability distributions, are summarized in the following subsections.

The aircraft operates by flying a set of missions. The bearing component is expected to operate under different loads that depend on the flight regime and, consequently, on the mission. Therefore, current mission, being part of the aircraft-level ODD, is also part of the ML constituent ODD. Duration of flight regimes in different missions follow a Gaussian distribution (with different parameters across different regimes). Condition indicators for the bearing component, provided by HUMS, are also expected to follow specified probability distributions. The aircraft is expected to operate under different environment conditions. All these dimensions frame the ODD of the ML constituent. Their descriptions, measurement units (if applicable), data types and sources are summarized in ***Table 1***. Numerical values for ODD dimensions (ranges, categorical values, distribution parameters) are available in ***Table 2***.

ML constituent ODD does **not** include abnormal loads, abnormal (unexpected) CI values, unknown flight regimes and missions, unexpected environment. These should be verified by robustness verification methods. In general, adverse inputs should be prevented from entering the ML model. For that, runtime monitoring techniques can be employed (discussed in Section 2.2.3.2).

Table 1. *ML constituent ODD dimensions (AvS = Avionics Software; G ~ m/s²).*

Input name	Description	Unit	Data type	Source
Ball energy	Condition Indicator (CI)	G^2/Hz	Numeric (float)	HUMS
Cage energy	Condition Indicator (CI)	G^2/Hz	Numeric (float)	HUMS
Inner race energy	Condition Indicator (CI)	G^2/Hz	Numeric (float)	HUMS
Outer race energy	Condition Indicator (CI)	G^2/Hz	Numeric (float)	HUMS
Shaft order 1	Condition Indicator (CI)	G	Numeric (float)	HUMS
Shaft order 2	Condition Indicator (CI)	G	Numeric (float)	HUMS
Shaft order 3	Condition Indicator (CI)	G	Numeric (float)	HUMS
Torque	Aircraft Parametric Data	%	Numeric (float)	HUMS
Current regime	Current flight regime (flight phase)	n/a	Categorical (string)	AvS
Nominal load	Nominal load of the component. Each flight regime has a different nominal load.	n/a	Numeric (float)	AvS
Current mission profile	Information about current mission of the vehicle: sequence of flight regimes, duration in each regime.	n/a	Regimes: Categorical (strings) Durations: Numeric (integers)	AvS
Environment	Condition, in which the mission is flown.	n/a	Categorical (string)	AvS

Table 2. ML constituent ODD ranges.

Dimension name	Limits	Comment
Ball Energy	Weibull distribution ($\lambda = 1.29E-04, k = 0.419$)	λ – scale, k – shape
Cage Energy	Weibull distribution ($\lambda = 9.09E-04, k = 0.242$)	Same as above
Inner Race Energy	Weibull distribution ($\lambda = 7.69E-05, k = 0.549$)	Same as above
Outer Race Energy	Weibull distribution ($\lambda = 8.73E-05, k = 0.387$)	Same as above
Shaft Order 1	Weibull distribution ($\lambda = 0.705, k = 3.59$)	Same as above
Shaft Order 2	Weibull distribution ($\lambda = 0.280, k = 2.056$)	Same as above
Shaft Order 3	Weibull distribution ($\lambda = 0.970, k = 2.666$)	Same as above
Torque	Min: 0; Max: 160	
Current regime	One of the following: Ground, Takeoff, Ascent, Forward Flight, Descent, Hover, Land	
Nominal load	One of the following values: • 2.1 (Ground) • 9.2 (Takeoff, Land) • 8.2 (Ascent) • 4 (Forward Flight) • 7.6 (Descent) • 7 (Hover) Nominal load is the mean of the Gaussian distribution with the following parameters (**NOTE:** only positive values are allowed): • Ground: $\mu = 2.1$ and $\sigma = 0.5$ • Takeoff: $\mu = 9.2$ and $\sigma = 1.5$ • Ascent: $\mu = 8.2$ and $\sigma = 1.5$ • Hover: $\mu = 7$ and $\sigma = 1.0$ • Forward Flight: $\mu = 4$ and $\sigma = 1.5$ • Descent: $\mu = 7.6$ and $\sigma = 1.5$ • Land: $\mu = 9.2$ and $\sigma = 0.5$	There is no easy way of measuring current (actual) component load during operation. Statistically, in each flight regime, actual load shall be within $\pm\sigma$ of the provided probability distribution. **NOTE:** Higher loads may occur during a flight regime if there is an abnormal maneuver (e.g., banked turn).
Current mission	Sequence of mission regimes – one of the pre-defined patterns. **Example of a mission pattern**: *Ground – Takeoff – Ascent – Forward Flight (Short) – Descent – Land*. **Flight regime duration:** Within $\pm\sigma$ of the Gaussian distribution with • Ground: $\mu = 10$ and $\sigma = 5$ • Takeoff: $\mu = 1$ and $\sigma = 0$ • Ascent: $\mu = 10$ and $\sigma = 5$ • Hover (Short): $\mu = 5$ and $\sigma = 3$ • Hover (Long): $\mu = 10$ and $\sigma = 10$ • Forward Flight (Short): $\mu = 50$ and $\sigma = 20$ • Forward Flight(Long): $\mu = 180$ and $\sigma = 20$ • Descent: $\mu = 10$ and $\sigma = 5$ Land: $\mu = 1$ and $\sigma = 0$	Full list of mission patterns is not provided here for brevity.
Environment condition	One of the following: desert, normal (non-desert)	

2.2.3.2 ML constituent ODD monitoring aspects

Correct function of the RUL estimator (ML constituent) shall be guaranteed inside its operational design domain. During operation, unexpected/adverse inputs may occur, for example, due to abnormal maneuvers of the aircraft, unexpected environment phenomena. While ML model's behavior for such inputs shall be evaluated via robustness verification methods, an appropriate risk mitigation for preventing adverse inputs and possible unintended behavior of the model is the use of runtime monitoring in the ML constituent, in particular, the ML constituent ODD monitoring.

In case of the RUL estimator, some checks on the quality of certain model inputs, such as condition indicators, shall be performed outside of ML constituent by the HUMS using *data quality indicators*. As further elaborated in Section 5.4.3.1, such checks can detect, for example, fluctuations in the CI values over multiple time steps in the time window, flag the incoming data correspondingly, and block it from entering the ML constituent (in this case no RUL prediction will be provided).

Other checks for out-of-ODD inputs can be implemented as monitors inside the ML constituent to perform, for example, range checks for inputs, which have min/max values prescribed by the ML constituent ODD (e.g., Torque), correctness of one-hot encoded categorical inputs (e.g., exactly one category is equal to "1" at every time step), and out-of-distribution checks. Further discussion and investigation of ODD monitoring, in particular, of out-of-distribution detection, is out of scope of the ForMuLA report.

2.2.4 Requirements

Provided lists of requirements are not complete and have exemplary purpose for the discussion and demonstration of formal methods carried out in the ForMuLA IPC.

This section provides information about the functional requirements both at the level of the RUL estimator function and at the system level. It also describes the main categories of non-functional requirements for the ML constituent.

2.2.4.1 Functional requirements

Table 3 provides several functional requirements for the aircraft/vehicle (VH) and its maintenance that are related to the RUL function of the Vehicle Health System (VHS). A selected list of system-level functional requirements for the VHS (ML system) is shown in ***Table 4***. These requirements can be refined into more detailed requirements for the RUL ML constituent, provided in ***Table 5***.

Table 3. *Selected aircraft (vehicle)-level requirements related to RUL.*

ID	Requirement
VH-1	The vehicle shall operate in two different environments: desert and non-desert (normal), which differently affect the degradation of the mechanical bearing component.
VH-2	The vehicle shall execute a set of predefined flight mission types. Each mission type (pattern) is an ordered sequence of flight regimes.
PHM-1	The vehicle shall provide means for estimation of RUL of the bearing component on ground, i.e., during flight preparation.

Table 4. *Selected ML system-level requirements (VHS).*

ID	Requirement
VHS-1	The VHS shall provide a function for estimation of RUL of the bearing component.
VHS-2	The VHS RUL function shall accept as inputs statistical condition indicators for the bearing component, current flight regime, nominal load in the current flight regime, current environment condition, and current mission.

VHS-3	The VHS RUL function shall compute the RUL desertic and non-desertic environments.
VHS-4	The VHS shall accept a sequence of input snapshots within a time window. The last snapshot in the time window shall correspond to most recent timestamp.
VHS-5	The VHS shall store snapshots for current and preceding time steps in memory and update it at every time step.
VHS-6	The VHS RUL function shall operate inside its operational design domain.
VHS-7	The VHS RUL function shall have a maximum admissible error of +30% (RUL over-estimation) and -10% (RUL under-estimation) in the "normal" range, i.e., when bearing component health state is not low/critical.
VHS-8	The VHS RUL function shall have a maximum admissible error of +5% (RUL over-estimation) and -15% (RUL under-estimation) in the "critical" range, i.e., when bearing component health state is low or critical – greater prediction accuracy is required for degraded component to avoid incorrect decisions for component inspection and maintenance.
VHS-9	The average absolute error of the RUL function shall not exceed 15 hours.

Table 5. *Selected functional requirements for the RUL ML constituent.*

ID	Requirement
RUL-ML-1	The ML constituent shall return a numerical value corresponding to the predicted remaining useful life of the bearing component in hours.
RUL-ML-2	The ML constituent inputs, encoded as numerical features, shall include statistical condition indicators for the bearing component, current flight regime, nominal load in the current flight regime, current environment condition, and current mission.
RUL-ML-3	The ML constituent shall have a categorical feature related to current environment. Its domain shall include two values: desert and non-desert.
RUL-ML-4	The ML constituent shall accept multivariate time series data as input.
RUL-ML-5	The time series data shall be organized as a two-dimensional array, where rows represent consecutive time steps and columns represent input features.
RUL-ML-6	The time step for the time series data shall be equal to 60 minutes.
RUL-ML-7	The number of time steps in the time series data array shall be equal to 40.
RUL-ML-8	The ML constituent shall update the input and perform inference with the new input every 60 minutes.
RUL-ML-9	The ML constituent shall ensure correct function within the input ranges specified by the ML constituent ODD.
RUL-ML-10	The over-estimation error of the ML constituent shall not exceed 30% in the "normal" range.
RUL-ML-11	The under-estimation error of the ML constituent shall not exceed 10% in the "normal" range.
RUL-ML-12	The over-estimation error of the ML constituent shall not exceed 5% in the "critical" range.
RUL-ML-13	The under-estimation error of the ML constituent shall not exceed 15% in the "critical" range.
RUL-ML-14	The "critical" range shall correspond to the last 100 hours of component RUL; the "normal" range shall correspond to all hours before the last 100 hours.
RUL-ML-15	The ML constituent average RMSE on the test dataset shall not exceed 15 hours.

The main *performance* requirement for the RUL estimator is the *accuracy* of the estimation (percentage of admissible error) that can be measured by comparing the estimation accuracy at each input point in the test dataset to the ground truth. Metrics, such as RMSE, can also be applied to quantify the error [18]. Respective requirements include the bound on over-estimations and under-estimations of the RUL, in particular, in the "critical zone". RUL estimation accuracy also has safety considerations, as elaborated in Section 2.2.7.

2.2.4.2 Non-functional requirements

In ***Table 6***, a selected list of non-functional requirements for the RUL estimator is provided. They include stability and monotonicity of the estimator (formal definitions of these properties can be found in Section

3.4), as well as the impact of the operating environment on the RUL prediction. These requirements can be formalized as properties and verified using formal methods, as demonstrated in Section 5.4.

Table 6. Selected non-functional requirements for the RUL ML constituent.

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. This value is estimated from available degradation data for the bearing.
RUL-ML-Stab-2	For a perturbation of a single condition indicator at a single time step within any input time window in the ML constituent ODD, the output deviation of the RUL estimator shall not exceed **10 hours**. The max perturbation value for which the requirement must hold corresponds to RUL-ML-Stab-1. Requirement applies to each condition indicator.
RUL-ML-Stab-3	For a simultaneous perturbation of all condition indicators at a single time step (e.g., due to a resonance frequency) within any time window in the ML constituent ODD, the output deviation of the RUL estimator shall not exceed **10 hours**. The maximum perturbation value for which the requirement must hold corresponds to RUL-ML-Stab-1.
RUL-ML-Mon-1	For an increased growth rate of a single condition indicator (may occur, for example, when a particular failure/damage occurs in the bearing, which increases its degradation) within any input time window in the ML constituent ODD, the estimator shall output a non-increasing value of the RUL. Requirement applies to each condition indicator.
RUL-ML-Mon-2	For an increased growth rate of all condition indicators (may occur, for example, due to simultaneous development of a number of failures or due to excessive load) within any input time window in the ML constituent ODD, the estimator shall output a non-increasing value of the RUL.
RUL-ML-Env-1	RUL estimator shall predict a smaller RUL for a desert environment than for a non-desert environment, all other inputs being equal. *(A desert operating environment has higher impact on the bearing degradation than a non-desert environment)*

2.2.5 Data description

Presented use case does not represent any concrete product of Collins Aerospace. ForMuLA project only used synthetic data from simulations, both for training and testing, because real data from the field was not immediately available during IPC execution for the selected application and mechanical component. The main goal of ForMuLA is the analysis of applicability of FM as means of compliance for the assurance and certification objectives for ML constituents and practical demonstration on a use case, not the verification or certification of a final product. Therefore, synthetic data was sufficient for the project. However, in general the importance of using real data, in particular for testing, is highly acknowledged.

The data for the RUL use case is in a form of multivariate time series[10] that describe the degradation of bearings installed in the same type of assembly (drivetrain). Data is related to a specific type of aircraft (rotorcraft) and specific type of sensor, which is a single axis piezoelectric accelerometer mounted on the outside of the gearbox near the bearing. The time series start at a healthy state and end at a failure state of the bearing. Additional sequences may be provided that either do not start at a 100% healthy state or end earlier than the component fails.

Currently available data comes from simulations. Collins Aerospace possesses physics-based bearing degradation models that can be configured to run the data collection process under a set of simulation scenarios. Following model parameters can be varied: component and cross-component degradation coefficients, environment factor (imitates different operational environments; affects the degradation rate),

[10] In the remainder of this report, they are referred to as degradation sequences that capture run-to-failure conditions of the component.

which allows to collect data for different types of bearings installed in different mechanical assemblies, as well as model the uncertainties (e.g., manufacturing, installation) within the component. These simulation parameters have associated probability distributions. Additional parameters can be introduced in the simulation model to consider other important types of uncertainties, for example, aberrations and/or aging of sensors that monitor the component.

Each simulation is traced to a simulation scenario. The latter consists of a sequence of flight missions, while every mission, in turn, includes a sequence of flight phases. Each flight phase has a duration described by a Gaussian distribution (see ***Table 2***). To generate a new executable scenario, a sequence of missions is randomly sampled from a set of predefined mission patterns (see example of a mission pattern in ***Table 2***). The scenario is then simulated to obtain a new run-to-failure sequence. Altogether, these sequences form the dataset that undergoes data preparation activities (e.g., normalization, labeling, feature engineering) and is split into training, validation and test sets.

Summary of bearing datasets to be used in the RUL use case of the ForMuLA IPC is provided in ***Table 7***. There is a number of degradation sequences. Each such sequence has samples corresponding to subsequent time steps. Each sample is a snapshot of inputs (e.g., condition indicators) at a given time step. Samples are labeled with RUL values at current time step. The average number of samples in the sequences is also provided.

Table 7*. Summary of the available bearing degradation data.*

Item	Value
Number of degradation sequences	100 (more can be generated)
Number of features	21 (some are categorical)
Number of categorical features	3
Average sequence length (steps)	631
Time step duration	1 (flight) hour
Missing or wrong data entries	None
Data preparation toolchain	Available: Includes feature engineering, labeling, normalization, split into training/validation/test datasets.
Metadata	Available: Includes simulation scenario (for traceability), simulation model version and timestamp, random seed, environment condition.

2.2.6 ML model description

The ML model is trained offline using available time series data and a supervised learning method.

To achieve an accurate RUL prediction at current time step, the snapshot of inputs (e.g., CIs, flight phase, current component load) taken at this single step is often not sufficient. Instead, as discussed in Section 2.1.1, RUL estimation functions typically accept a sequence of inputs, also called a *time window* [16]. The last row in this window is the current time step, i.e., the step at which the RUL is being estimated, while all preceding rows are "historical" (preceding) time steps. This suggests a 2-dimensional (2D) input structure, with the first dimension being the number of time steps in the window, and the second dimension being the number of features. Based on the available bearing degradation data (number of condition indicators and other features) and considering that categorical features need to be one-hot encoded, the total number of input features at each time step is **40**. Similarly**, 40 hours** is the reasonable size of the time window validated with Collins Aerospace SME (each step is 1h, therefore, the number of time steps is 40 as well) needed to get an accurate RUL prediction. Altogether, the number of inputs to the ML model, i.e., the cumulative number of entries in the time window for all time steps, is on the order of 1000.

Given the complexity of the input space, a deep learning solution has been selected for the RUL estimator, namely, a convolutional neural network (CNN). The choice of CNN is justified with the fact that this type of neural network is capable of automatically extracting features from a large number of raw inputs, thus

reducing or completely removing the need of manual feature engineering. This is particularly relevant to images, where raw data is represented by pixels, and crafting interpretable features from pixel data is often impractical. Two-dimensional input to the RUL estimator is similar to the image input representation. Moreover, despite the fact that certain inputs, such as CIs, are precomputed outside of ML constituent based on sensor data (i.e., these features are not purely "raw"), higher-level features for RUL may include degradation trends. Domain experience may not be sufficient to define such trends and consider them in predictions. Such trends could be automatically extracted by a CNN.

Neural network architecture of the RUL estimator is adapted from [17] and is summarized in ***Table 8***. As discussed above, it accepts as input a sequence of time steps. A number of convolutional layers is used to apply one-dimensional convolutions along the time sequence direction, thus extracting trends in separate features. These trends are then merged together via a fully connected layer. Activation functions at all layers are Rectified Linear Units (ReLUs). Dropout is used to mitigate overfitting. The CNN performs a *regression* task and outputs a numerical value, which is the predicted RUL value.

***Table 8**. Summary of the CNN architecture for the RUL estimator.*

Item	Value
Input size	two-dimensional; 40x40 window
Output size	1
Model type	convolutional neural network
Model task	regression
Number of convolutional layers	4
Type of convolutions	one-dimensional convolutions
Types of layers	convolutional, fully connected
Type of activations	ReLU
Total number of layers	12
Total number of learnable parameters	94500
Dropout probability	0.1

2.2.7 Safety considerations

Provided lists of failure conditions are not complete and are provided for exemplary purpose to support the discussion and formal methods demonstration carried out the ForMuLA IPC.

Rationales in EASA NPA 2022-03 guidance on (d) VHM (vibration health monitoring) system safety requirements [19] have been considered to derive the proposed classification. For an actual certification project associated assumptions would be listed in the functional hazard analysis and validated during the project.

Remaining useful life estimation function can be subject to various functional failures that may affect the safety, therefore, the use case is safety relevant. This is illustrated in ***Table 9***, where a set of representative failures of the RUL function, corresponding failure effects and the classification based on severity of the failure conditions effects (MAJ = Major; MIN = Minor) has been captured.

Table 9. Representative list of RUL use case failure conditions classified by hazard levels.

ID	Failure description	Failure effect	Class	Rationale
FC1	Undetected loss of RUL function combined with a critical degradation of the bearing or an actual failure.	MRO or Pilot mistakenly considers the component RUL "sufficient" to execute the flight mission without an inspection or maintenance action (in reality the component may develop a failure).	MAJ	Loss of RUL function may lead to both under-estimation and over-estimation of the RUL. In the latter case, there is a possible safety impact, i.e., lack or delay in critical decision making: pilot/MRO does not know that the component may fail soon and starts the flight mission – large reduction in safety margins.
FC2	Non-monotonic variation of RUL	Same as above	MAJ	Critical inspection of the component may be mistakenly skipped (especially if the variation is near the decision threshold)
FC3	Frozen value of RUL	Same as above	MAJ	Critical inspection of the component may be mistakenly skipped (especially if the variation is near the decision threshold)
FC4	Undetected loss of runtime monitoring of ML constituent ODD	Same as above	MAJ	Out-of-ODD input value may not be detected due to runtime monitor failure. Correctness of ML model outputs for out-of-ODD inputs may not be guaranteed, therefore, an undetected incorrect RUL prediction may appear (e.g., an overestimation), which may lead to skipping a critical inspection of a component that lead to an unexpected failure.
FC5	Use of outdated (untimely) inputs	Same as above	MAJ	Out-of-date inputs may reflect some previous component state with less degradation compared to the actual state. Consequently, the system may predict the RUL that is higher than real one, which may lead to skipping a critical inspection of the component.

3 Formal methods technologies for machine learning

This chapter provides necessary background on formal methods, including main definitions and high-level application categories. It then discusses property specifications and formal methods technologies that have been adapted to or specifically developed for machine learning. In addition, the chapter offers a discussion of other categories of methods, such as statistical methods, that can complement the analysis for some assurance objectives where formal methods are not applicable or face scalability limitations.

3.1 What are formal methods?

Formal Methods (FM) are typically defined as mathematical techniques used to develop software systems and verify their correctness. In commercial aviation, the use of formal methods in the development of software-based aerospace systems is allowed through the ED-216/DO-333 [1] supplement of the ED-12C/DO-178C [5] standard. ForMuLA report refers to the following definition of formal methods taken from DO-333:

> *Formal methods are mathematically based techniques for the specification, development, and verification of software aspects of digital systems. The mathematical basis of formal methods consists of formal logic, discrete mathematics, and computer-readable languages. The use of formal methods is motivated by the expectation that, as in other engineering disciplines, performing appropriate mathematical analyses can contribute to establishing the correctness and robustness of a design. For example, formal methods, because of their mathematical basis, are capable of:*
>
> • *Unambiguously describing requirements of software systems.*
> • *Enabling precise communication between engineers.*
> • *Providing verification evidence such as consistency and accuracy of a formally specified representation of software.*
> • *Providing verification evidence of the compliance of one formally specified representation with another.*

Possible applications of formal methods span across the entire development lifecycle of a software system, including (1) *formal specification* of the system and its requirements using different mathematical formalisms (e.g., first-order logic, finite state machines) that are used by effective reasoning tools; (2) support for *system development activities*, such as design exploration and architecture/program synthesis, and (3) *formal verification* that aims to provide formal proofs of correctness of intended algorithms, programs, and systems. Verification is the largest area that includes a number of traditional FM applications[11]. For example, *model checking* provides a sound, complete, and automatic verification method for finite-state models of software and hardware against specifications by exhaustive exploration[12]. *Proof assistants* are able to produce reliable proofs of mathematical theorems, often in an interactive fashion. *Static program analysis* techniques perform a direct and automated analysis of programs without executing them (for example, this is often used in compilers). ForMuLA report focuses on the use of formal methods for machine learning, thus leaving out of scope the detailed discussion of aforementioned traditional approaches. We refer an interested reader to FM survey works, such as [20].

Remark: *Formal methods are traditionally associated with rigorous and exhaustive analyses. To explore a broader scope of techniques for assurance and verification of ML, ForMuLA also considers* ***statistical methods****. The use of statistical methods is intrinsic to the design assurance process for ML as possible means of compliance for such objectives as data quality and ML model generalization. They can also be applied to property verification of ML models, for example, to mitigate scalability issues of exhaustive analyses of traditional FM, while increasing the thoroughness of the analysis.*

[11] Of course, both development and verification applications of FM are informed by a formal specification.
[12] Extensions exist to address the tractability of the analysis (for example, bounded model checking, symbolic mode checking).

3.2 Formal methods main definitions

This section provides a set of key definitions related to formal methods. Some definitions (e.g., soundness, completeness) are provided in the context of property verification,. However, they similarly apply to other applications of FM.

Property. A property is a mathematical statement, *i.e.,* a declarative sentence which is either true or false. A *formal method* can be used to determine whether a given property is valid or invalid.

For example, let f be a function $X \to Y$, X and Y being, respectively, the input space (domain) and the output space (codomain) of f. Also, let both input and output of f be two-dimensional, i.e., $X \in \mathbb{R}^2$ and $Y \in \mathbb{R}^2$. For a given subset of input points $X' \subset X$, the following property P can be formulated[13]:

$$P: \forall x \in X', \ \ f(x) \in Y' = \{(y_1, y_2) \in \mathbb{R}^2 | \ y_1 \leq y_2\} \tag{3.1}$$

where $Y' \subset Y$ is a region of the output space Y. An equivalent formulation is given below:

$$P: f(X') \subseteq Y' \tag{3.2}$$

where $f(X')$ is the *image* of X' by f, i.e., the set of all elements of the output space $Y \in \mathbb{R}^2$ that correspond to the output of f when applied to input points in X'. Formally, it is defined as $f(X') = \{f(x) \mid x \in X'\}$. Image $f(X')$ is often referred to as the set of outputs reachable from the inputs X', or *output reachable set*.

A property is *valid*[14] (resp. *invalid*) if it evaluates to True for each input x in X' (resp. False for at least one input x in X'). An illustration of a valid property P_1 is shown in ***Figure 6***. Here, the output reachable set $f(X_1')$ is fully contained in the region (half-space) $Y' = \{(y_1, y_2) \in \mathbb{R}^2 | \ y_1 \leq y_2\}$, therefore, the property is valid *w.r.t.* the input subset $X_1' \subset X$. Instead, an invalid property P_2 is illustrated in ***Figure 7***. Here, part of the output set $f(X_2')$ is out of the required region, i.e., certain inputs from $X_2' \subset X$ lead to outputs that violate the condition $y_1 \leq y_2$, thus invalidating P_2.

A property is *satisfiable* (SAT) if there exists at least one input that makes it evaluate to True. The property is *unsatisfiable* (UNSAT) in the opposite case.

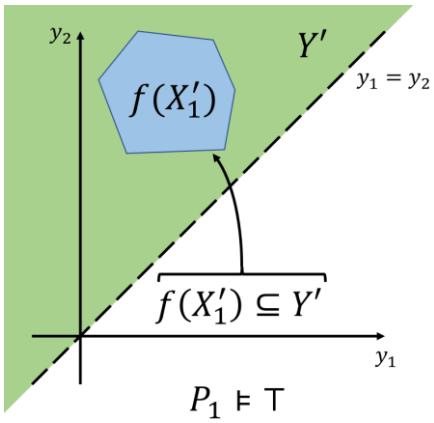

***Figure 6**. An example of a valid property.*

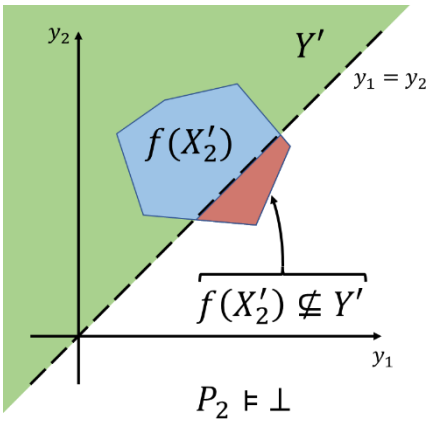

***Figure 7**. An example of an invalid property.*

[13] This example of property is a universal ("for all") statement. It is supposed to be true for *all* members of a set, which is expressed through a universal quantifier. Such properties are most common, and the remainder of this report focuses on such universal statements (i.e., *"for all inputs…",* rather than *"there exists some input such that…"*).

[14] Commonly used synonyms: true (false) property, the property holds (does not hold). Mathematically, valid property is written as $P \vDash \top$ ("P entails True") and invalid property is written as $P \vDash \bot$ ("P entails False").

Counterexample. Consider a property P, which is a universal statement. A *counterexample* (CEX) to P is an instance $\tilde{x}$ in X' for which the negation $\neg P$ is true. If there exists a counterexample to P, then P is invalid. Inversely, if there exists no counterexample to P, then P is valid.

An illustration of a counterexample to an invalid property P_2 is given in ***Figure 8***. P_2 is as in the previous example, i.e., P_2: $f(X_2') \subseteq Y' = \{(y_1, y_2) \in \mathbb{R}^2 | \, y_1 \leq y_2\}$. A counterexample is an element $\tilde{x} \in X_2'$ such that $\neg P_2$ is true, *i.e.,* such that $f(\tilde{x}) \notin Y'$.

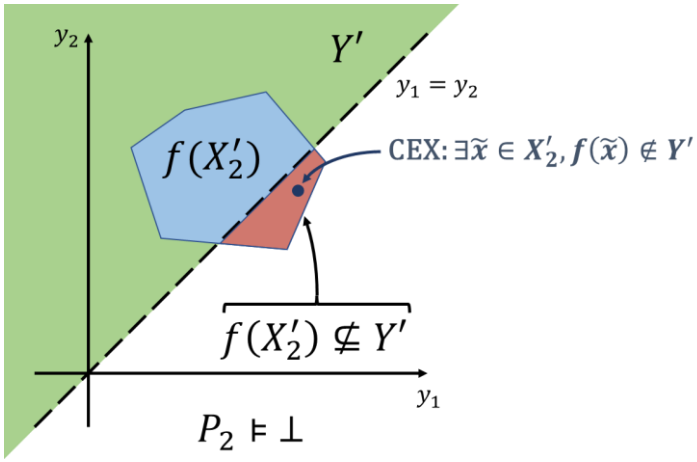

Figure 8. *A counterexample to an invalid property.*

Falsification. The falsification approach is a common way of property verification using formal methods, that aims at finding a counterexample for the property. To do so, the negation of the property ($\neg P$) is considered. If a formal method can identify at least one input from the input (sub)space specified for P that makes this negation evaluate to True, this, consequently, disproves P. This is because the universal statement in P becomes false, i.e., the property does not hold for all required inputs. Instead, if the verification problem for $\neg P$ has no solutions then P is declared valid.

To reason about the validity of properties, FM may rely on *approximations*. For example, rather than reasoning on an exact property formulation P: $f(X') \subseteq Y'$, the method may consider an approximation $\tilde{P}$: $\tilde{f}(X') \subseteq Y'$, where $\tilde{f}(X')$ approximates the set $f(X')$. Approximations in FM have practical value as they can be less computationally expensive to verify than the original property. However, considerations on method soundness exist, discussed below.

Soundness. A verification method is *sound* if for any property P it returns that P is valid **ONLY IF** P is valid. In other words, a sound method never has a *missed violation*, i.e., an invalid property is never declared valid.

Remark: *Missed violations are often referred to as false negatives. In this case, "negative" is an answer to the falsification problem, i.e., "Is there some point that violates the property?". A negative answer, i.e., "No, there are no points that violate the property", means that the property itself is valid. Therefore, false negative means that the property has been mistakenly declared valid, that is, a property violation has been missed.*

If a sound method relies on an approximation $\tilde{P}$ of P, then this must be a *conservative approximation*: $\tilde{P} \Rightarrow P$. In this case, if the method can prove that $\tilde{P}$ holds and returns True, then necessarily P is valid. For instance, a conservative approximation of P: $f(X') \subseteq Y'$ could be a property $\tilde{P}$: $\tilde{f}(X') \subseteq Y'$ where $\tilde{f}(X')$ is a superset of $f(X')$ *i.e.,* such that $\tilde{f}(X) \supseteq f(X)$. ***Figure 9*** illustrates an output set $f(X_1')$ of the function f and its conservative approximation $\tilde{f}(X_1')$ for a valid property P_1. It can be seen that the approximation subsumes the "real" output set, which shows its conservativeness. In this case, if the approximation meets the output constraint $y_1 \leq y_2$, it can be concluded that P_1 is valid. ***Figure 10*** illustrates that when a property is invalid (P_2 on the figure), the conservative approximation $\tilde{f}(X_2) \supseteq f(X_2)$ captures the invalidity as well.

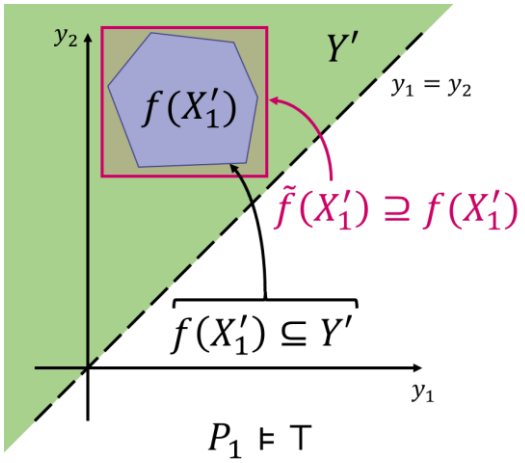

Figure 9. An example of a conservative approximation for a valid property.

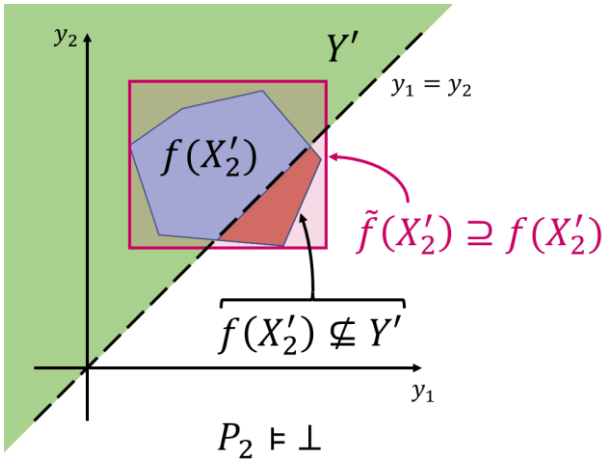

Figure 10. An example of a conservative approximation for an invalid property.

Completeness. A verification method is *complete* if for any property P it returns that P is valid **IF** P is valid. In other words, if a solution to the falsification problem exists, it will always be found by a complete method[15]; a *false alarm*[16] is never raised, i.e., a valid property is never declared invalid.

An example of a false alarm is provided in ***Figure 11***. It illustrates a valid property P_3: $f(X_3') \subset Y'$: all possible outputs $f(X_3')$ of the function f belong to the correct half-space $Y' = \{(y_1, y_2) \in \mathbb{R}^2 | \, y_1 \leq y_2\}$. The figure also shows the output of a sound method that deals with a conservative approximation of P_3, $\tilde{f}(X_3')$, where $\tilde{f}(X_3') \supseteq f(X_3)$, as a red rectangular area. The bottom right part of this conservative over-approximation $\tilde{f}(X_3')$ lies outside of Y'. Therefore, such method may declare P_3 invalid as it intersects with an undesired region, while in fact the property is valid. The method may also return a counterexample, i.e., some input from the region $\tilde{f}(X_3') \setminus Y'$ that is a *spurious* (misleading) *counterexample* for verification, i.e., a false alarm.

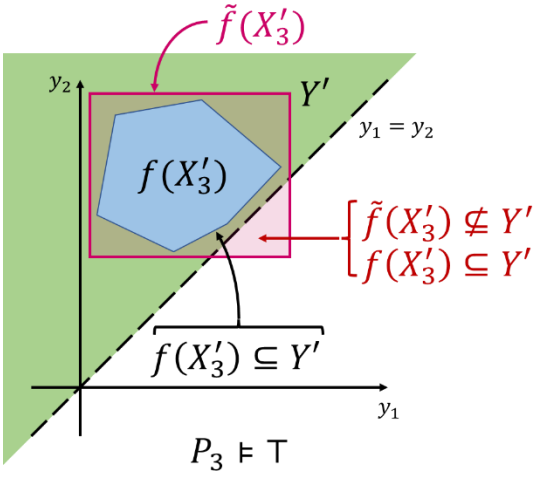

Figure 11. An example of a false alarm (false positive) for a valid property.

Soundness is a mandatory requirement for a formal method. Some FM are sound and complete, i.e., they can correctly prove or disprove *any* property that can be expressed in the formalism used by the method. Such methods do not rely on approximations and provide maximum precision of the analysis. Other types of

[15] Note that no assumption is made on the method scalability, i.e., even if the solution is guaranteed to be eventually found, this may not be done in reasonable time. Scalability is a typical problem for FM, especially for complete methods.
[16] Often referred to as a "false positive".

FM are instead *sound but incomplete*. Such methods typically use conservative approximations to preserve soundness, while they trade off completeness to reduce the computational burden. As discussed above, they may not be able to prove property violations. To avoid returning false alarms, they typically return an "unknown" answer when they determine that the property is invalid. This is because the invalidity is with respect to an approximation, which means that the property may in fact be valid if the exact analysis without approximation is performed.

3.3 High-level application categories of formal methods

Possible uses of formal methods span across specification, development, and verification. While *analysis* is considered the main FM activity (various formal analyses for systems and software exist, including ML), they can also perform *synthesis* functions, i.e., generate artifacts that can be used during development and verification. Generally, FM applications belong to one of the three following areas:

Property Inference. In the absence of, or in addition to, known properties to be verified, it may be possible to automatically infer characteristics of a model behavior, either related to parts of the model or to the model as a whole. In other words, this is a synthesis activity for the properties. For example, this is the case for neural networks, where techniques have been developed to extract layer properties or relationships between NN input and output.

Property Verification. Formal analyses can be employed to provide evidence that a property of interest is valid on a given model. The applicability of this approach is constrained: it depends on the extent to which the property is formalizable and verifiable from a theoretical and practical perspective. On the other hand, the formal model has to be a *conservative* representation of the original artifact to guarantee that, if the property holds for the model, then it holds for the artifact as well.

Automated Test Generation. Manual testing is an expensive and time-consuming activity. Many different methods have been proposed in the literature to automate the generation (synthesis) of test cases, according to the availability of a model for the artifact under test and to the chosen testing criteria: random and adaptive random testing, search-based testing, combinatorial testing, scenario-based testing, structural coverage-based testing, and others.

3.4 Property specifications for machine learning

Formal methods can be used to verify properties of machine learning models. As per the state-of-the-art, existing FM technologies and tools, in particular those for neural networks, only address a specific type of property formalization that associates a desired or a forbidden output region (or class) to a given input region. Such formalizations are referred to as *input-output relationships* or *input-output properties*[17]. This section provides an overview of key properties that are relevant to machine learning. For a more detailed overview the reader can consider [21] or [22].

The scope of ML properties can be either local or global. A **local property** is defined for a given input point $x \in X$ or a subset of points $X' \subset X$ of the input space X. That is, local properties must hold for *some* specific inputs. A **global property** is defined over the entire input space X of the ML model. Global properties must hold for *all* inputs.

[17] Sometimes, also the term *reachability property* is used. The rationale is that formal analysis computes, possibly with approximation, all possible ML model outputs that are *reachable* from (can be the result of ML model computation for) a given set of inputs.

3.4.1.1 Generic input-output relationship

Properties of ML models are typically expressed over their inputs and outputs, without involving internal model structure[18]. This is because application experts can use domain knowledge to require certain output behavior of the ML model based on certain inputs, while the internal organization of the model is often a black box to them (e.g., model architecture or learnable parameters, such as weights and biases). A generic form of these properties is expressed as an *if-then* relationship:

$$P(x) \Rightarrow Q(y), \tag{3.3}$$

where $P(x)$ is a precondition on the input x (premise), and $Q(y)$ is a postcondition over the output y (consequent). Both precondition and postcondition can be expressed differently depending on the type of x and y. For example, for a numeric input a precondition may impose a range (or a multi-dimensional subspace if multiple inputs are considered), while for a categorical value a set of admissible values (e.g., classes) may be specified. Arithmetic and logical relationships between the inputs can also be imposed by the precondition. The same applies to the outputs and the postcondition.

Let $\hat{f}$ be an ML model that approximates some function $f: X \to Y$. For the example below, let $X = \{(x_1, x_2) \in \mathbb{R}^2\}$ and $Y = \{(y_1, y_2) \in \mathbb{R}^2\}$. An input-output property can be exemplified as follows:

$$(x_1 \geq 0) \wedge (x_1 \leq 3) \wedge (x_1 \geq x_2) \Rightarrow (y_1 < y_2)$$

This property requires the first output y_1 of the ML model to be strictly less than its second output y_2, given that the input x_1 is in the interval $[0, 3]$ and is greater than or equal to x_2.

At the "base" (lowest) level all ML model properties discussed below are expressed as input-output relationships. They specify desired output behavior based on a set of constraints over the ML model input.

3.4.1.2 Stability

Following the definition of ML model stability[19], this type of properties limits the admissible deviation of the ML model output, given a bounded perturbation of its inputs. Stability properties are defined for perturbations in *normal operating conditions*, that is, perturbations over the inputs inside the ML constituent ODD. Input perturbation is bounded by a value often referred to as δ (delta). Similarly, the maximum admissible deviation of the output, such that the output can be still considered "expected" or "correct", is often denoted as ε (epsilon). This results in the following "delta-epsilon" formulation of stability properties:

$$\|x' - x\| < \delta \Rightarrow \left\|\hat{f}(x') - \hat{f}(x)\right\| < \varepsilon, \tag{3.4}$$

where $x \in X$ is the original input belonging to the input space X of the ML model, $x' \in X$ is the perturbed input, $\hat{f}(x)$ and $\hat{f}(x')$ are ML model outputs for, respectively, x and x', δ and ε are as discussed above ($\delta, \varepsilon \in \mathbb{R}_{>0}$), and $\|\cdot\|$ is a norm that measures the distance between original and perturbed inputs and outputs. The

[18] In general, it is possible to define properties that also involve internal structure of ML models (e.g., hidden layers of a neural network) if they have meaningful semantics and can be traced to some system/model requirements. In particular, *explainable AI* methods could help to understand the behavior of internal elements of models and to make use of these elements for improved traceability and richer specifications. Formal methods can also be used to infer properties from datasets and ML models, as discussed in Sections 3.5.1.1 and 4.3.1-4.3.4.

[19] Note the distinction between *stability* and *robustness* of the ML model, where the former has the scope of only normal operating conditions, while the latter subsumes it and considers both normal and adverse conditions (that is, stability, as well as edge cases, adversarial cases, etc.). Current academic literature on the verification of neural networks does not make any distinction between the two terms and only uses "robustness" to describe the properties.

right-hand side of Equation (3.4) suggests a regression output. Instead, if $\hat{f}(\cdot)$ returns a class, then a stability property shall impose that the class does not change in the presence of an input perturbation[20]:

$$\|x' - x\| < \delta \Rightarrow \hat{f}(x') = \hat{f}(x). \tag{3.5}$$

If input/output perturbations, i.e., δ and ε, are relative values (e.g., "*an input perturbation of 1%*") then the following form would apply for a given input point x:

$$\forall x' \in X: \|x' - x\| < \delta\|x\| \Rightarrow \left\|\hat{f}(x') - \hat{f}(x)\right\| < \varepsilon\,\hat{f}(x). \tag{3.6}$$

One can observe that the formulations above contain an implication (⇒). Therefore, as discussed above, they establish relationships between ML model inputs and outputs.

Local stability. A property that captures the stability of the ML model around a given input point is a local stability property: a perturbation of a concrete input shall result in a slight or no change in the output of the model (e.g., bounded error in the case of regression or no change in prediction class in the case of classification). Local stability properties are accepted by the majority of FM tools for ML, e.g., tools for neural networks verification.

Global stability. A more general formulation is global stability property, that states that for *any* input point from the ML model's input space ($\forall x \in X$) the property formulated as in Equation (3.4) must be valid. Global stability can also be expressed as a bound on the ratio between the change in the output and the change in the input. This is a notion of Lipschitz continuity [23] discussed in Section 3.4.1.4.

Perturbation measurement. Input perturbations (and, respectively, deviations in the ML model outputs) can be quantified in different ways using different types of norms. For example, the L_1 norm (also known as Manhattan distance) and the L_2 norm (Euclidean distance) are different ways of measuring the distance between the two inputs or outputs. The infinity norm (L_∞) that records the greatest perturbation magnitude among all input elements is also widely used for measuring perturbations. The norms are defined as below:

$$L_\infty: \left|\left|\boldsymbol{x} - \boldsymbol{x}'\right|\right|_\infty = \max_i |x_i - x_i'| \tag{3.7a}$$

$$L_1: \left|\left|\boldsymbol{x} - \boldsymbol{x}'\right|\right|_1 = \sum_i |x_i - x_i'| \tag{3.7b}$$

$$L_2: \left|\left|\boldsymbol{x} - \boldsymbol{x}'\right|\right|_2 = \left(\sum_i |x_i - x_i'|^2\right)^{\frac{1}{2}} \tag{3.7c}$$

3.4.1.3 Robustness

As discussed above, in general ML model robustness captures both the stability in normal operating conditions, with respective property defined as in Section 3.4.1.2, as well as the capability of the ML model to not exhibit unintended behavior in the presence of adverse inputs, such as the ones outside of the ML constituent ODD, as well as edge/corner cases, adversarial cases, and out-of-distribution cases. There is no specific formalization of a robustness property that is different from the one defined above (Equation 3.4)[21], therefore, in this section we focus on certain variations of this formulation that can be used for identification of adversarial examples, which can be considered adverse inputs.

[20] Typically, raw outputs of a classification model are probabilities or scores of different classes. Therefore, for classification models, stability properties can also be formalized similarly to (3.4) imposing that the score of some class shall deviate by no more than ε .

[21] It is a matter of input region where the property is defined; if the region is outside of ODD or near the ODD boundary, then the same stability property may be referred to as "robustness property", because it considers inputs outside of the normal range.

Some FM technologies, such as the ones based on optimization (e.g., MILP) can be employed to search for inputs that are "challenging" for the ML model. For example, such inputs may be located close to a decision boundary of the model, such that a small perturbation would change the output class, or near an unknown discontinuity of the function, such that the function output may deviate significantly in the neighborhood of an input. These challenging inputs may be used both as tests and as dataset augmentations to improve ML model robustness and mitigate the risks of adversarial attacks. Several formulations of the optimization problem can be used. The first one aims at minimizing the input perturbation. To express it, the delta-epsilon formulation in Equation (3.4) can be modified as follows:

$$\begin{aligned} &\min_{\delta} \mu(\delta) \\ s.t.\ &\delta \in \Delta \\ &\exists (x, x'): \|x' - x\| < \delta \Rightarrow \left\|\hat{f}(x') - \hat{f}(x)\right\| \geq \varepsilon \end{aligned} \tag{3.8}$$

Here, μ is a cost function defined for the perturbation, and Δ is a set of possible perturbations. The formulation aims to find a minimum-cost perturbation for the output deviation to exceed the bound of ε. For instance, in the context of adversarial attacks, using a smallest/cheapest perturbation helps the attacker to stay undetected. Another formulation aims at maximizing the loss:

$$\begin{aligned} &\text{argmax}_{\delta} \left\|\hat{f}(x') - \hat{f}(x)\right\| \\ s.t.\ &\delta \in \Delta \\ &\|x' - x\| < \delta \end{aligned} \tag{3.9}$$

This formulation focuses on finding a worst-case error (the "most incorrect" output) of the model given a set of possible input perturbations.

3.4.1.4 Lipschitz continuity

Lipschitz continuity is a global property, characterizing the behavior of the ML model over its whole input space. It consists of a *Lipschitz constant* which measures the sensitivity of the model to input perturbations:

$$\left\|\hat{f}(x') - \hat{f}(x)\right\| \leq \boldsymbol{\theta}\|x' - x\|, \tag{3.10}$$

where $\|\cdot\|$ is a norm that measures the distance between original and perturbed inputs and outputs, e.g., L_1, L_2 or L_∞ norm, as in Equations (3.7a) - (3.7c), and θ is the Lipschitz constant. In other words, the constant is an upper bound on the ratio between the variations of the outputs and the variations of the inputs of an ML model $\hat{f}$ (more generally, of some function f). The smaller the constant θ, the more robust is the ML model with respect to perturbations.

As shown in [24], neural networks with low Lipschitz constants offer better generalization capabilities together with stronger robustness against adversarial attacks. Thus, it is of major interest to enforce and to demonstrate the existence of such low Lipschitz constant.

3.4.1.5 Monotonicity

In certain applications, the function that is represented by the ML model is required to exhibit monotonic behavior. For example, given a monotonic change in some input feature(s), the output of the model should also change monotonically, i.e., increase or decrease. Monotonicity is a typical requirement for regression models of various kinds, such as those that perform monitoring of degradation conditions of various components, in particular, PHM applications.

One possible definition of monotonicity of the ML model, adapted from [25], could be the following. Let $\hat{f}: X \to Y$ be an ML model that approximates the function $f: X \to Y$, and let S be the set of inputs of $\hat{f}$ (input features). The output of the model $\hat{f}$ is *monotonically increasing* in features $S' \subseteq S$ if and only if each feature

in S' is totally ordered and for any two inputs $x, x' \in X$ that are (1) non-decreasing in features $S', \forall i \in S'\ x[i] \leq x'[i]$, and (2) are equal in all other features $k \in S \setminus S'$: $\forall k, x[k] = x'[k]$, the output of the model $\hat{f}$ is *non-decreasing*: $\hat{f}(x) \leq \hat{f}(x')$. From this definition, a global monotonicity property for monotonic increase can be defined in the following way:

$$\forall (x, x') \in X\ \ (\forall i \in S'\ x[i] \leq x'[i]\ ) \wedge (\forall k \in S \setminus S'\ x[k] = x'[k]\ ) \Rightarrow \hat{f}(x) \leq \hat{f}(x') \tag{3.11}$$

A similar formulation can be created for a monotonically decreasing (non-increasing) model output.

It is also possible to define local monotonicity properties, which may be more amenable to verification using FM, given the complexity of the global formulation. A local property is formulated in the neighborhood of a given point x in the input space X of $\hat{f}$. In case of monotonicity, given a monotonic "shift" (change) in selected features from the input point $x \in X$, a monotonic change (increase or decrease) in the output of $\hat{f}$ is imposed.

As an example, consider a model $\hat{f}$, with a one-dimensional input and output spaces $X \in \mathbb{R}$ and $Y \in \mathbb{R}$, where the output is expected to monotonically decrease with the increasing value of the input. Given an input point x, one can define following local properties:

$$\forall\, x' : x\ \leq\ x' \leq x + \delta \Rightarrow \hat{f}(x') \leq \hat{f}(x) \tag{3.12a}$$

$$\forall\, x' : x - \delta \leq x' \leq x \Rightarrow \hat{f}(x') \geq \hat{f}(x) \tag{3.12b}$$

where x' is an input point in the neighborhood of x, to which a monotonic shift δ has been applied. Equation (3.12a) is a *forward decreasing monotonicity* property stating that for any monotonic increase bounded by δ, the output must be non-increasing. Equation (3.12b) is a *backward decreasing monotonicity* that requires that, locally to the point x, decreasing value of the input shall lead to a non-decreasing output.

Limited non-monotonicity. An additional term ε can be added to the right-hand side of the equations to account for non-monotonic behavior that is *admissible*. In this case, the properties impose that the output is allowed to have a *limited* change in the direction that is opposite to the expected one (but not an unlimited growth/decrease); such properties are locally applicable to show a bounded deviation from the desired behavior. The addendum ε turns the forward and backward decreasing local monotonicity properties, shown in Equations (3.12a)-(3.12b), into *limited increasing monotonicity* properties:

$$\forall\, x' : x\ \leq\ x' \leq x + \delta \Rightarrow \hat{f}(x') \leq \hat{f}(x) + \varepsilon \tag{3.13a}$$

$$\forall\, x' : x - \delta \leq x' \leq x \Rightarrow \hat{f}(x') \geq \hat{f}(x) - \varepsilon \tag{3.13b}$$

Example of non-monotonic regions of $\hat{f}$ are shown in ***Figure 12***. Function $\hat{f}(x)$ is shown in orange color, while the blue line represents an ideal decreasing monotonic behavior with increasing x and is shown for reference. There are two non-monotonic regions. In the first (left) one, for a bounded increase x'_1 of the input x around the point x_1 ($x_1\ \leq x'_1 \leq x_1 + \delta$), there is an increase of $\hat{f}$ that exceeds the admissible value of ε, therefore, the limited increasing monotonicity is violated around the point x_1. Instead, for the second (right) region, a bounded increase x'_2 from the point x_2 ($x_2\ \leq x'_2 \leq x_2 + \delta$) leads to only a slight increase in $\hat{f}$ (less than ε), hence, it does not violate the property.

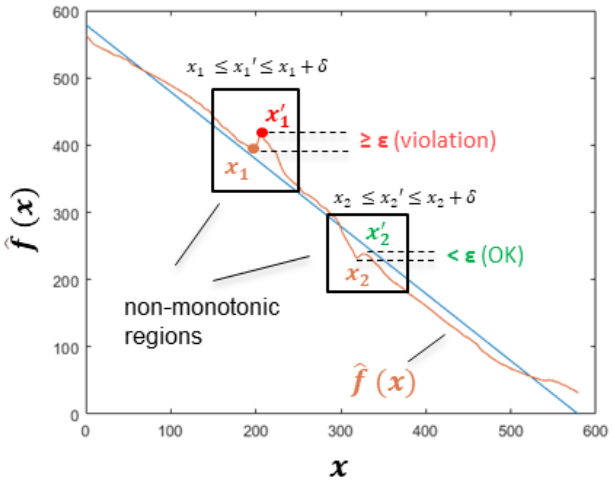

***Figure 12**. Example of valid and invalid monotonicity properties in non-monotonic regions of the function.*

3.4.1.6 Other types of properties for ML

Additional property types may also be considered for some types of ML models. For some input spaces, *equivalence classes* can be defined, each of them containing certain object types. These classes can be used to specify *invariance properties*, for example, enforcing that certain transformations of an object on an image (scaling, rotation, etc.) shall lead to the same output of the classifier. Such properties, sometimes also referred to as semantic invariance properties [26], are domain specific. For models that contain a state, such as recurrent neural networks, *temporal specifications* could be provided using conventional formalisms, such as temporal logics. Temporal behaviors can also be exhibited in reinforcement learning applications. The reader is referred to [26] for more details.

3.5 Formal methods technologies applied to machine learning

Current section provides an overview of existing FM-based technologies and tools applicable to or specifically developed for machine learning. A significant part of this overview focuses on *property verification* which is the main FM application in ML design and V&V. Interested readers may refer to [27] and [28] for more detailed discussions on formal verification tools. In addition, certain FM technologies also find their applications in ML development processes [27].

At present moment, neural networks (NN), in particular deep learning, is the most popular technology in ML. They have gained significant interest in the aviation industry due to their remarkable performance at solving complex problems, with potential applications in safety-critical avionics systems [29] [30]. In safety-critical contexts, assurance and verification of NNs is crucial. According to the state-of-the-art, *verification of neural networks (VNN)* is currently the main focus of formal methods applications to ML [27] [31]. For this reason, the FM technology overview in this report focuses on VNN. However, other FM approaches applicable to broader areas/types of machine learning are also discussed when this is relevant.

3.5.1 Complete formal methods

Complete formal methods refer to algorithms and solvers that are both *sound and complete*: they can precisely report whether a given property holds on a model, generally providing a counterexample in case the property does not hold. Such methods do not miss property violations stating that a property holds on a

model while in reality it does not[22]. Furthermore, they also prevent *false alarms* (false positives), i.e., they state that the property is violated only if it is indeed so.

Completeness is not always achievable from a theoretical perspective. In practice, when achievable, it comes with soundness at a price of high execution time. Neural network verification problems are NP-hard problems[23], therefore, sound and complete VNN algorithms face scalability issues. In the current practice, complete FM generally do not scale to large NN architectures. For VNN, execution time may grow exponentially with the increase of either the NN complexity (number of layers, number of neurons, different types of activation functions) or the property complexity (e.g., larger input perturbations increase the complexity of robustness properties), or both. Complete FM are often also limited to certain NN architectures and activations, for example, neural networks with piecewise linear activation functions, such as ReLU.

3.5.1.1 SMT-based

Satisfiability Modulo Theories (SMT) [32] solvers aim at automatically determining the satisfiability of a formula within a given theory, such as Boolean logic or linear arithmetic, or within a combination of theories. For example, an SMT solver can determine the satisfiability of the formula $(x \geq 2) \wedge (y \geq 3) \wedge (x + y \leq 6)$ within the linear real arithmetic theory. Here, the solver will return that the formula is satisfiable, together with an instance that satisfies the formula, e.g., $(x = 2.5, y = 3)$. At their core, SMT solvers, such as Z3 [33] or CVC5 [34], use SAT (Boolean satisfiability) algorithms that are complemented by dedicated theory solvers. An example of a theory solver is a simplex algorithm used to reason about linear real arithmetic theory.

SMT-based formal methods have been developed to verify or infer properties of ML models [27]. The literature has mainly focused on SMT-based approaches for **verification of neural networks** [35] [36] [37]. Such approaches reduce the NN verification problem to a constraint satisfaction problem by encoding the neural network and the negation of the property of interest as a set of linear arithmetic and logical constraints over the variables that represent the inputs and the outputs of NN elements (neurons). A solution to the constraints, if it exists, represents a counterexample to the property, i.e., the property is violated when this CEX is used as input. Otherwise, if the constraints are unsatisfiable, then no CEX exists, and the property is valid. For reasoning on neural network models, dedicated theory solvers have been proposed. One example is the Reluplex method [35], which is based on the simplex algorithm extended to support constraints that describe ReLU activations. State-of-the-art VNN tools, such as [35] and [38], have been developed using this method. They are currently limited to piecewise linear activation functions, such as ReLU, and to certain types of NN layers, such as fully connected, max- and average pooling, and convolutional layers.

SMT-based FM have also been developed to **verify decision tree models**. A *decision tree* is a type of ML that can perform both regression and classification. Except the leaves, each node of a decision tree branches either to the left or to the right subtree, depending on a Boolean condition over the input. The leaves of the decision tree correspond to the labels that can be assigned to the input (for classification) or values such as real numbers (for regression). Different formal methods approaches have been developed to reason about decision trees and their ensembles (e.g., random forests) [27]. Among them, [39] uses SMT for the verification of an input-output property over a decision tree ensemble. Additionally, [40] introduces a SMT-based tool called VeriGB (Verifier of Gradient Boosted models) which can verify stability properties of gradient boosted decision trees.

One critical obstacle to the verification of neural networks is the lack of definition of their intended behavior. Sometimes, NN functional requirements are “hidden” in the datasets, i.e., they are not available in explicit

[22] Note, however, that soundness is not typically guaranteed with respect to floating-point arithmetic, but only with respect to computations on real arithmetic that ignores rounding errors and may thus differ from the actual computations.
[23] Problems that are “at least as hard as the hardest problems in the NP space” (NP stands for “non-deterministic polynomial time” class of computation problems; more details can be found, e.g., in https://en.wikipedia.org/wiki/NP-hardness).

form. Hence, comprehensive specifications for verifying neural networks are often missing. Recently, SMT-based FM approaches have been proposed to perform **property inference from trained neural networks**. One approach, introduced by Gopinath et al. [41], allows an automated discovery of input-output properties. The method works on DNN with ReLU activation functions and uses the SMT-based Reluplex backend mentioned above. These properties are of the form $\boldsymbol{P(x) \Rightarrow Q(y)}$, as in Equation (3.3); both the precondition $P(x)$ and the postcondition $Q(y)$ must be convex, respectively, over the NN inputs x and NN outputs y. The inference of such properties relies on the notion of an *activation pattern*, which is a partial mapping from the set of neurons to their ReLU activation statuses (on, off). The idea is to discover a minimal activation pattern that yields a given postcondition $Q(y)$. Similarly, the approach can infer inner layer properties. Such properties are of the form $\boldsymbol{\sigma_l \Rightarrow Q(y)}$, where $\boldsymbol{\sigma_l}$ is the neuron activation status at layer $\boldsymbol{l}$. Neural network property inference using formal methods can have applications in learning assurance and explainability, as further discussed in Section 4.

Important information about the intended behavior of the ML model can be explicitly present in the datasets. Recently, the use of SMT has been proposed for **property inference from datasets.** For example, in [42], a non-greedy iterative algorithm with SMT solving is used for automatic extraction of rules from data that contains feasible and infeasible examples. It is able to infer mathematical expressions with Boolean logic and linear arithmetic over the rationals, where each expression must be valid (resp. invalid) for all feasible (resp. infeasible) examples in the data. Current limitation is the assumption on data accuracy, i.e., absence of noise in the labels. However, methods for extracting rules from noisy data have also been proposed (e.g., see [43]).

SMT-based methods have also been employed in **guiding the learning process** for neural networks, which allows the NNs to be correct-by-construction with respect to certain properties. For example, in [25] a *counterexample-guided training* procedure is employed to iteratively train a new NN instance during the training phase and to identify *monotonicity counterexamples*, i.e., inputs that violate the monotonicity property (see definitions in Section 3.4.1.5) of a feedforward neural network during verification. These counterexamples are then added to the training dataset to improve monotonicity of the function at the next training iteration. An optimization extension of the SMT technology, Optimization Modulo Theories (OMT) [44], is used for this purpose.

In the same work [25], SMT/OMT is also used to **construct a safety envelope** for a neural network, namely, the monotonicity envelope. The method can be used to construct the envelope on-the-fly at prediction time with minimum overhead. Such envelope can be used to enforce the monotonicity of the output at runtime. In case if non-monotonic output of the NN is identified, it is overridden by the monotonicity envelope value.

3.5.1.2 MILP-based

A Mixed Integer Linear Program (MILP) is an optimization problem involving both real and integer decision variables, and linear constraints over these variables. State-of-the-art MILP solvers can find solutions to optimization problems with thousands of variables and constraints using the branch-and-bound algorithm and its extensions, the linear relaxation technique to quickly estimate lower and upper bounds of the optimal solution, as well as advanced heuristics to improve performance. MILP is a sound and complete method.

MILP-based techniques and tools have been developed for the **verification of neural networks**. Similarly to SMT, neural network models can be reduced to a MILP via a number of different encodings, such as the ones proposed in [45] and [31]. The key challenge is to capture the behavior of the ReLU activation function for NN elements. So far, piecewise linear activation functions, such as ReLU, have been the only type of activations supported by sound and complete MILP-based FM[24] [27]. For example, the encoding in [31] uses

[24] Other activation functions, such as $tanh$, are also supported by some tools (e.g., [101]) via piecewise linear approximations. With these approximations MILP-based tools are sound (but not complete).

the traditional big-M method [46] to capture the branching in the ReLUs. Several MILP-based VNN tools exist, such as [45], [47] and [48].

Similarly to SMT, MILP-based FM have been used to **guide the learning process** and obtain correct-by-construction neural networks with respect to certain properties, such as monotonicity. For example, the authors of [49] propose a procedure that *iteratively* trains a neural network with heuristic monotonicity regularizers, verifies the trained model monotonicity using MILP (NNs with piecewise linear activations are supported), and tightens the regularization term in the objective function until the NN passes the verification. Numerical evaluation is performed on shallow networks, but an extension to deep learning is also discussed.

MILP has been employed to **estimate Lipschitz constants** in neural networks with ReLU activation functions under l_1 or l_∞ norms. A MILP-based formulation and a corresponding algorithm are discussed in [50].

Being an optimization-based method, MILP also supports analyses related to model robustness described in Section 3.4.1.3, such as **adversarial example generation** (finding minimal perturbations to violate certain property, identifying inputs maximizing the effect of an attack).

Mixed integer linear programming has also been employed for **inference of properties from the data.** Similarly to the SMT-based approach discussed above, the key idea is to formulate the problem as a MILP that, using a *grammar* (structure of expected rules to be found), aims at finding the rules that (1) comply with the grammar, (2) satisfy all feasible ("positive") examples in the data and (3) violate all infeasible ("negative") examples in the data. Following this idea, the method in [51] is capable of producing linear, quadratic and trigonometric constraints. The authors of [52] propose an iterative algorithm to improve the scalability of the inference.

3.5.1.3 Reachability analysis

Reachability analysis aims at computing the set of reachable outputs of an ML model for a given set of inputs. For instance, a set of inputs, as well as corresponding set of reachable outputs, can be represented as polytopes, or some other structure[25]. Sound and complete **verification procedures** with reachability analysis are available for **neural networks**. For example, the approach in [53] leverages structures called *star sets* to represent input and output sets of NN layers. Affine transformations can be efficiently computed for star sets. Starting from the NN input layer, they are propagated through inner layers, estimating their reachable output sets, all the way to the output layer. The final output set can be checked for intersection with the negation of the property of interest to check its validity (no intersection means that the property holds, and vice versa). In other words, one can analyze whether the output set contains some undesired/unexpected outputs that invalidate the property. The verification method is complete for ReLU activations, but involves a lot of branching, because each possible combination of ReLU activation statuses produces a distinct star set for each neuron, therefore, their number can grow rapidly leading to scalability problems for large NNs.

Interval arithmetic (also known as interval analysis or bound propagation) is an efficient technique for rigorously estimating the output range of a function from its input range, and can be seen as a type of reachability analysis. It works by computing and propagating the output intervals for each operation in the function. The technique can be used to evaluate bounds of the NN output. Typically, initial bounds of the outputs of NN elements are a coarse abstraction that may result in a significant over-approximation of the output interval, thus preventing the property validity check (*unknown* answer from the solver). Completeness of the analysis can be achieved by iterative refinement, i.e., iteratively tightening the bounds via abstraction refinement on symbolic intervals. A number of state-of-the-art VNN methods, such as [54], [55] and [56], are based on such symbolic interval analysis approach and offer various improvements and extensions of the

[25] Depending on representation, reachability analysis can be either complete or incomplete. In the latter case, an over-approximation of inputs and outputs is used, i.e., abstract interpretation (see Section 3.5.2.1).

method to scale the formal analysis. For example, these methods are known to outperform SMT-based FM tools on the aerospace benchmark ACAS XU [57] by orders of magnitude in terms of execution time.

3.5.1.4 Other complete formal methods

Many complete verifiers for neural networks, such as, for example, [58] and [37], rely on the traditional **branch and bound method**. They explore the entire search space by (1) recursively splitting the original NN verification problem into subproblems (e.g., by splitting each neuron piecewise into positive and negative outcomes of the ReLU activation), i.e., *branching* and (2) using some incomplete verifier to estimate a best possibly achievable solution for each subproblem, i.e., *bounding*. A large body of recent work focuses on proposing such incomplete verifiers that can compute more precise bounds for NN elements in a faster way.

Other types of encodings and solvers have also been proposed for sound and complete verification of neural networks, such as *quadratic programming* and *semidefinite programming*. An interested reader is referred to comprehensive technology overviews in [28], [27] and [31] for additional information.

3.5.2 Incomplete formal methods

Incomplete formal methods trade off completeness of the analysis for an improvement in scalability. In case of verification of neural networks, these methods typically outperform complete FM by orders of magnitude in terms of analysis time for NNs with thousands of neurons, as reported in the recent benchmarking results of some state-of-the-art tools [59]. Incomplete methods are sound[26], but suffer from false alarms (false positives). This is often caused by approximations that these methods use in computations, which often makes it unclear whether a property valid. When a violation is detected, a counterexample produced by such methods may be *spurious*, that is, it may represent an input that in practice does not violate the property. Incomplete FM are generally less limited to specific neural network architectures and activations. For example, they support a variety of activation functions beyond ReLU. Also, incomplete FM approaches have been recently proposed for verification of recurrent neural network (RNN) models [60][27].

3.5.2.1 Abstract Interpretation

Abstract interpretation is a verification approach that uses approximation and abstraction in a mathematical setting. It consists of an abstract domain (representation of input/output spaces, e.g., polyhedra, intervals, zonotopes), a pair of abstraction and concretization functions, and a sound abstract semantic function. In this approach, verification is still treated as a reachability problem, as described in Section 3.5.1.3, but it is solved by reasoning over an abstract system obtained from the reduction of the original model. Abstract interpretation methods are used in many VNN tools [53] [61] [62] [63], where they typically symbolize each node of the NN, apply an abstraction function over its computation, and eventually check whether the property is valid by considering the estimated output values of the NN after their concretization. In other words, starting from an abstraction of NN inputs, the method propagates this abstraction through NN layers by applying affine transformations and activation functions in the hidden layers, until it reaches the output layer, where the final abstraction represents an over-approximation of the output reachable set.

Several abstract domains have been proposed for neural networks. For example, along with exact analysis using star sets, the approach in [53] also provides sound over-approximations for ReLU, as well as other, non-piecewise linear activation functions. These abstractions represent the input set of the NN as a polytope. In the case of ReLU, instead of branching on each neuron to capture all possible activation combinations

[26] Often also with respect to floating point arithmetic.
[27] Note that formal verification approaches for RNN are currently seminal works and have low maturity. VNN community currently primarily focuses on verification of feedforward and convolutional neural networks. Therefore, technologies for RNN verification are not further discussed in the ForMuLA report.

(described in Section 3.5.1.3), the method accumulates all possible outputs in a single star set structure (corresponding polytope abstraction of ReLU is shown in ***Figure 13a***). The method is complemented with a dedicated star set based structure, called ImageStar [64], that is efficient for representing image data, thus supporting the analysis of convolutional networks.

In [62], several other abstract domains are proposed, both polytope and zonotope. Corresponding ReLU approximations are illustrated in ***Figure 13b*** (zonotope) and ***Figure 13c-d*** (polytope). These approximations are coarser *w.r.t.* [53], therefore, the method scales better to large neural networks, but this may result in larger amount of *unknown* answers (more properties remain unproven by the analysis).

Optimized abstractions have also been recently proposed [63]. Here, the coarseness of the abstraction can be controlled by several parameters, while the verification problem is formulated as an optimization problem aimed at finding the smallest abstraction that is sufficient to prove the property. Corresponding ReLU polytope abstraction is shown in ***Figure 13e***. Here, the slope of the lower bound is controlled by a parameter α that is optimized by the method. It can be seen that, as corner cases, when $\alpha = 0$ (resp., $\alpha = 1$), abstraction reduces to the one in ***Figure 13c*** (resp., ***Figure 13d***). Therefore, the method can provide more accurate analysis with respect to [62], yet possibly more expensive in terms of computation overhead.

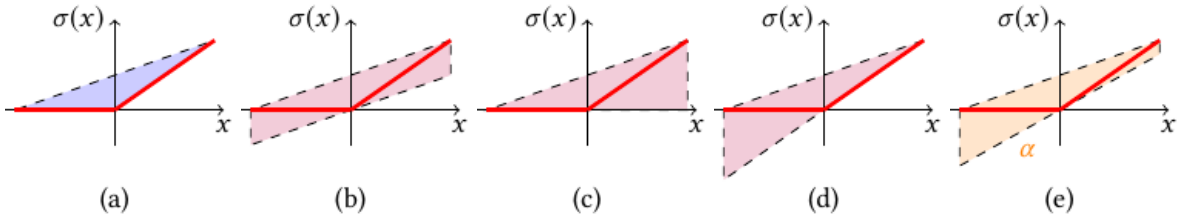

Figure 13*. Approximations of ReLU activation function.*

ReLU function is shown in red color on each subfigure. Approximations are shown with (a) polytope, (b) zonotope, (c-d) coarser polytopes, (e) polytope abstraction with optimizable lower bound.

Verification of other types of ML models is also possible with abstract interpretation. For example, incomplete FM approach for **verification of local robustness properties of Support Vector Machines (SVMs)** has been proposed in [65]. Similarly to neural networks, the input set of an SVM is propagated through the model, yielding an over-approximation of the reachable outputs. To this end, the method uses a zonotope abstraction together with a dedicated abstract transformer that soundly approximates the semantics of SVMs. In particular, it uses new abstract transformers for the most commonly used kernels of SVMs, e.g., it can verify SVMs with linear, polynomial and gaussian radial basis function (RBF) kernels.

3.5.2.2 Other incomplete methods

Incomplete FM often make part of complete methods. For example, incomplete approaches are used as background theory solvers to **estimate the bounds of the outputs of neurons** and entire layers of a neural network. This is a key step of the majority of VNN methods and it has to be implemented as efficiently as possible to reduce the overall analysis time. Various symbolic interval analysis and bound propagation methods mentioned in Section 3.5.1.3 have been developed and can be used both as standalone incomplete verifiers, as well as parts of complete verification methods (e.g., see [55], [66], [63]). One important direction for such incomplete approaches is their effectiveness on specialized hardware, such as GPUs, which can provide a significant boost to scalability of VNN tools (e.g., consider [67] and [68]).

A set of incomplete methods provide a sound **approximation of the Lipschitz constant**, i.e., an upper bound, under L_1, L_∞ and also L_2 norms [69] [70] [71]. They support not only ReLU but also sigmoid and tanh activation functions. Some of these approximate methods frame the Lipschitz estimation problem as a semi-definite program (SDP) [71].

3.5.3 Standard formats used in formal methods for ML

With the rapidly growing number of published VNN methods and tools, a standardized exchange format becomes important in order to reduce the effort for evaluating different tools in an industrial setup, as well as to enable efficient tool benchmarking. The Open Neural Network eXchange (ONNX) format [72] becomes a *de-facto* community standard for ML interoperability. Most of existing VNN tools provide support for ONNX, while major ML development frameworks (PyTorch, Keras/TensorFlow, Matlab) provide import/export functions to convert their internal formats to ONNX. Similarly, the VNN-LIB format [73] is a community effort to standardize the I/O format of VNN tools and benchmarks. It builds on the aforementioned ONNX format for model description, and on the Satisfiability Modulo Theories Library (SMT-LIB) format for property specification. VNN-LIB is currently supported by most state-of-the-art VNN tools. Another initiative is the DNNV framework [74], which gathers 13 state-of-the-art VNN tools and ease their use by proposing a unified I/O format. This format is the combination of an ONNX neural network, the related expected property in an expressive domain-specific language, and the name of the VNN tool to be used.

3.6 Scalability limitations of formal methods

As anticipated in Section 1.1.4, formal methods often target exhaustive analysis of the state space, therefore, they are typically computationally heavy and face scalability issues. Their applications to machine learning are not an exception. FM scalability is affected by several factors, among which the main ones are complexity of the model and complexity of the property. In the case of neural networks, model complexity can be measured, for example, in number of neurons, number of hidden layers, or activation function types. This is similar also for other ML models. Property complexity may be related to the size of the input space, covered by the property, number of conditions to be verified, and non-linearity of the constraints. Due to the fact that many FM-based analyses are related to solving NP-hard problems, one may often observe exponential growth in analysis/solving time with the increase in either model complexity, or property complexity, or both. This is particularly relevant for exact analyses discussed above, however, approximate (incomplete) methods are also subject to the same issues. The problem is further exacerbated by the fact that the number of properties may be large, for example, if one needs to define and verify properties on every point in a (potentially large) test dataset.

Scalability problems may lead to unreasonable analysis time, thus making the use of FM impractical. There is a wide body of recent academic research focusing on improving the scalability of formal methods for ML, where the most effort is focused on neural networks and deep learning. Existing VNN tools improve every year, while new techniques and tools also appear. Tool authors (mainly, academic research groups) tweak the core algorithms of their tools to support more challenging use cases and also develop various heuristics to speed up the analysis. ForMuLA IPC does not discuss algorithmic improvements of particular VNN (or other) tools, but rather provides an overview of some recent tool-agnostic approaches and activities that have the potential to boost the FM performance for ML. The overview below does not aim to be exhaustive, therefore, it is organized as a (non-exhaustive) list of examples.

Verification of close-to-output NN layers. The approach in [75] presents a scalable verification approach of safety properties of direct perception neural networks. It addresses the problems of formal specification of NNs and of scalability by (1) specifying the expected behavior of the NN through training an *input property characterizer* and (2) proposing a scalable verification which only considers the last layers (also called *close-to-output* layers) of the NN. The input property characterizer is itself a neural network that takes as inputs the outputs of the l-th layer of the original NN, with l close to L, L being the last layer. For instance, the characterizer could be trained to output *True* when the outputs of the l-th layer of the NN correspond to an input image where “the road strongly bends to the right”. As a result, the safety verification becomes a decision problem: decide if there exists an output of the l-th layer of the NN such that the input property characterizer network outputs *True* and the NN outputs an undesired value, e.g., “strongly steer to the left”.

This decision problem only involves the l-th, l+1-th, … L-th layers of the original NN and the input property characterizer NN, which dramatically reduces the size of the verification problem, making it amenable to FM analysis. Although the motivation of [75] is to verify direct perception neural networks, the approach also applies to any deep neural network for which input constraints are hard to characterize.

Proof reuse for continuous verification. Similarly to other analyses, there is an expectation that property verification completes in reasonable time, but the latter is defined differently based on application, business need, available budget, and so on. For example, when an ML model is trained only once and is then frozen without any further updates, several days can be allocated to complete rigorous property verification activities. This may not be acceptable if the model has to undergo frequent updates. ForMuLA IPC only covers *offline* learning when the inference model is frozen, i.e. not updated during operation. However, offline updates to the ML model with subsequent updates of the inference model could be possible. Such updates may include, for example, re-training due to availability of new data or fine-tuning of model parameters. In such cases, previously established verification results may no longer be valid. Property verification is often a time-consuming process, therefore, *continuous (re)verification* with FM may introduce significant overhead in the ML lifecycle. The authors of [76] suggest several approaches to minimize this overhead for neural networks by reusing previously generated proofs and verification results. The methods rely on reusing state abstractions, network abstractions, and Lipschitz constants at different NN layers to speed up formal verification of the updated NN with enlarged input space and/or modified network parameters.

VNN competition. The trend of using ML-enabled systems in different contexts, including safety-critical ones, has accelerated the research in the area of FM tools for VNN in the recent years. The VNN research community is currently setting up a “common ground” for benchmarking of new and existing tools in the form of a competition (VNN-COMP [59]). It intends to bring together researchers working on VNN techniques and invites both academic institutions and industries to participate either as tool providers or benchmark providers. Each year state-of-the-art VNN tools are evaluated on a set of challenging benchmarks. By the moment of publishing of the ForMuLA report, the competition has been held 3 times. Image benchmarks and computer vision applications are still the main focus of the competition, but the organizers also welcome benchmarks from other domains, including aerospace that has by far been represented by the ACAS XU system, as well as the Remaining Useful Life use case discussed in the ForMuLA report.

3.7 Statistical methods

A variety of formal methods technologies for machine learning has been developed. These technologies primarily target ML model verification, though they also find applications in other phases of the ML development lifecycle, as further discussed in Section 4. However, scalability is a typical barrier for using FM, for example, on complex ML models, such as deep neural networks. Sound and complete verification is often not achievable, whilst on certain models and/or properties of high complexity no formal verification result, even with approximate incomplete analyses, may be available in a reasonable timeframe (see discussion on complexity in Section 3.6).

Among alternative approaches, **statistical methods** are particularly relevant for supporting assurance activities. For example, in the absence of a formal guarantee, one could rely on setting up a statistical argument, such as checking a number of random trials to estimate the probability of property violation. Beyond that, these methods may be employed for analyses that are out of reach of FM, e.g., not amenable to formal verification or *a-priori* intractable for FM. While the main focus of ForMuLA is the role of formal

methods in the ML development lifecycle, this section provides a brief overview of relevant statistical approaches[28]. The practical use of some of these techniques is further demonstrated in Section 5.

Data quality assessment. Statistics is one of the key instruments in a data scientist's toolbox. In particular, they may provide effective means for verification of data quality requirements. A large body of work is available on the use of statistical methods for e.g., detection of anomalies and novelty in the data and independence testing (e.g., one can refer to [77] and [78]).

Data completeness with respect to requirements and ML constituent ODD is an important consideration to ensure the capability of the ML model to generalize to unseen data. Various statistical methods exist to assess dataset completeness. The basic idea is to divide the ML constituent ODD into subspaces of some form, by the granularity required for associated operating parameters, and to check that there is a sufficient number of "examples" (inputs) in the dataset that belong to each subspace. *Design of Experiment (DoE)* methods are typically used for this purpose (e.g., refer to [79] and [80]). DoE is also relevant for other activities, such as *synthetic data generation* and *test scenario generation*. Here, one needs to perform sampling of datasets/scenarios in a way that guarantees high coverage of the ML constituent ODD, e.g., by using Latin hypercube sampling, or another similar approach. For highly multi-dimensional data, DoE methods may become challenging due to high computational cost.

Another noteworthy application of statistical methods for data verification is the assessment of *data representativeness*, where the aim is to ensure that datasets contain examples that have been independently sampled from the input space and follow the right (expected) distribution. For low-dimensional data, if the expected distribution is known, statistical methods, such as goodness-of-fit tests (e.g., Chi-Square) can be applied to check data representativeness. For high-dimensional data, other techniques are required. One example is the generic framework called *distribution discriminator* that has been introduced in the CoDANN-1 report [3].

ML model generalization. The possibility of the ML model to generalize to unseen data is one of the key aspects of its trustworthiness. A traditional approach for assessing the generalization capability of the ML model is the use of the test (holdout) dataset. However, this would only allow to estimate the *in-sample* error of the model, while more rigorous guarantees may be required for assurance[29]. Relevant statistical methods (e.g., statistical learning theory techniques) are discussed in [3].

Property verification. Complete formal methods discussed in Section 3.5 guarantee that the property of interest will eventually be proven but make no assumption on termination and available computational resources. Incomplete FM sacrifice completeness to terminate faster, but due to approximations used in the analysis, may not be able to verify the property, i.e., an unknown answer may be provided. Additionally, they may also face scalability barriers on complex ML models and properties. When a formal guarantee cannot be provided, one may resort to statistical methods. For example, *falsification* may be performed by a *simulation-based method*, where several random inputs are sampled from the input space and corresponding outputs are checked against the property of interest. If at least one of them is violated, it is returned as a counterexample, and the property is declared invalid. For a "softer" (e.g., probabilistic) requirement on property validity, one can apply the Monte-Carlo method and compute some statistic on violations within a

[28] One can argue that being *mathematically based* techniques, statistical methods may also fall under the category of FM, as per definition in Section 3.1. However, ForMuLA refers to conventional use of the term "formal methods" in aviation (e.g., as defined by ED-216/DO-333), therefore, statistical methods in this report are discussed as complementary means of achieving certain assurance objectives that may also be (partially) addressed by FM.

[29] Basically, one needs to demonstrate that the model can generalize to *any* data input that follows the input distribution, not just to the examples that are present in the test dataset. The latter is only an approximation of the overall generalizability of the model.

given number of trials. A statistical argument about property validity can then be built by showing that the number of detected violations does not exceed some required percentage.

3.8 Hybrid verification procedures

Another mitigation for scalability issues of sound and complete FM-based approaches is a combination of methods. Several types of techniques can be used together to maximize thoroughness of verification, while ensuring termination of the analysis in reasonable time. Resulting *hybrid verification procedures* can include, for example, multiple FM tools (e.g., one complete and one incomplete) invoked in sequence or in parallel, or a combination of FM and statistical methods. This may be especially relevant when the number of properties to be verified for the ML model is high. Even if verification of a single property instance may be reasonable by some complete FM (in terms of computation time), the total verification time could easily become impractical. This is further discussed and exemplified in Section 5.4.

Following are examples of possible hybrid verification procedures:

1. Properties are verified with an incomplete FM to obtain a formal proof of validity for as many properties as possible. The properties, for which the method returns an "unknown" answer, have been neither proved nor disproved. As a second step, they could be verified with a complete FM. Overall execution time can be significantly reduced due to "pre-solving" by an incomplete method.
2. For complex ML models and/or properties, instead of using an exact (complete) verification method, unknown properties could be verified by a simulation-based method in an attempt to falsify them [53]. If no counterexample is found, the property remains unknown (or a statistical argument can be made, as discussed in Section 3.7).
3. Several FM tools could be run in parallel.
4. If completeness is the priority, one may start instead from a complete FM (exact verification) and resort to incomplete FM or statistical approaches if verification time exceeds the threshold. Similarly, timeout could be used in other combinations of methods, e.g., invoking a complete method for some limited time in case neither incomplete methods nor statistical methods were able to verify the property.

Remark: *Some FM tools already incorporate several different verification options, or a combination of steps for solving the verification problem (e.g., LP, MILP and heuristics) within the same tool. In principle, these can also be considered hybrid.*

Verification at system level. Hybrid procedures can also be employed to facilitate verification of properties at ML system/subsystem level. One example is the neural network based controller ACAS Xu [57] where NNs aim at avoiding a collision between the two aircrafts (ownship and intruder). Here, verification of properties at the level of ML model (e.g., see [35]) may not suffice to demonstrate that the safety requirement is met. Instead, it may be necessary to verify properties at *system* level, e.g., "*for all the possible initial states of the intruder and ownship, no collision shall happen*". In this case, FM can be used in combination with *simulation* techniques to reason about the property and decide whether it holds or not [30].

4 Applications of formal methods specific to ML

This section discusses FM applications that are specific to machine learning and can address existing challenges in learning assurance. These applications aim at supporting the training process, as well as at improving ML model robustness and generalization capabilities. Furthermore, novel algorithms and tools developed in the recent years are extending the applicability of traditional formal methods also on ML models, allowing to increase confidence in ML model functional correctness, and to provide new means supporting model optimization and development explainability. Throughout Chapter 4, the process steps on which formal methods are proposed to be applied are linked to the associated objectives from the W-shaped learning assurance process that was introduced in the EASA Concept Paper for Level 1&2 ML[30].

4.1 Formal methods for supporting the learning process

This group of FM applications aims at contributing to the learning process, from the improvement and verification of the datasets and of the learning algorithm stability, to the assessment of ML model generalization capability in specific cases. Certain applications discussed in this section are amenable to analysis using statistical methods[31].

Figure 14 illustrates the allocation of FM applications of this group on the W-cycle diagram. Each symbol (square, triangle, diamond) refers to one of the subgroups discussed below. Each application is allocated to the W-cycle phase that it can support. For example, *inference of constraints from datasets* is intended to support the training process, therefore, this application is allocated to the Model Training phase on the diagram. Applications, for which the use of statistical methods is envisioned, are marked with an asterisk (*).

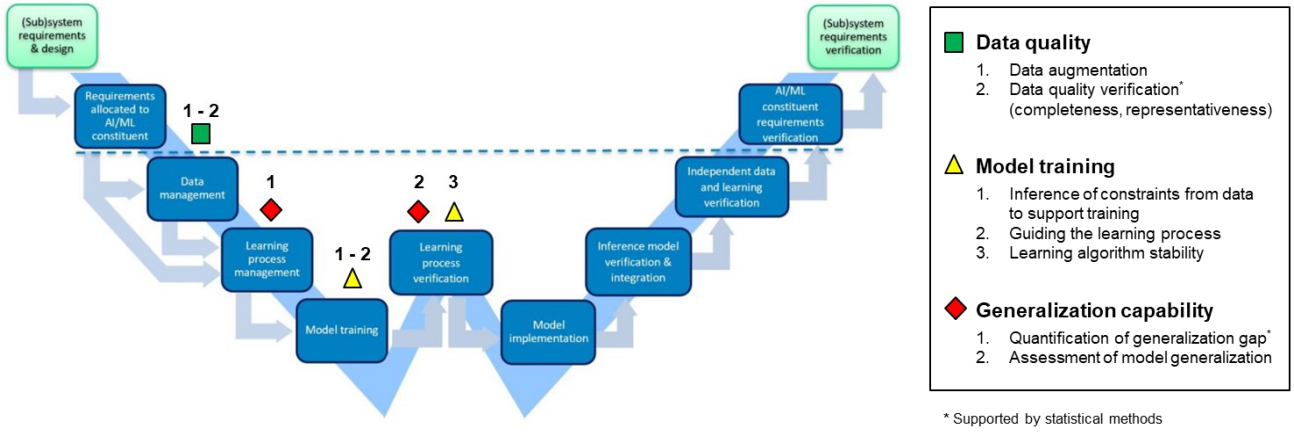

***Figure 14**. Applications of FM for supporting the learning process allocated on the W-cycle diagram.*

4.1.1 Data quality

Ensuring high quality of the data is at the heart of learning assurance processes and is specifically emphasized in the existing guidance. Certain objectives can be supported by formal methods. For example, FM can support **data augmentation**. In image recognition applications, new images can be obtained from existing ones by applying linear transformations that affect characteristics such as lighting and color, position and orientation of objects in space. These images are added to the training dataset. One of the goals of data

[30] Note that some FM applications discussed here cannot be directly mapped to and/or suggested as anticipated MoC for objectives described in the EASA AI Concept Paper [2]. However, all applications intend to support the development process and the quality of resulting ML models.

[31] As explained in Section 3.1, in this report selected statistical methods are discussed as complementary means of carrying out the analyses for certain assurance objectives, where FM are not applicable or face scalability limitations.

augmentation is to obtain higher coverage of the ML constituent ODD (see [81], [82], [83]) that should improve generalization of the resulting model after training. Formal methods can be employed here to automate techniques based on linear transformations, once the input space is mathematically defined.

To support **data quality verification**, one can use *statistical methods* discussed in Section 3.7. Both numerical and categorical features can be analyzed, for example, by clustering or binning values of particular features and analyzing how these groups are allocated along the expected value intervals prescribed by the ML constituent ODD. Such checks allow to assess **data completeness**. Well-known statistical metrics (e.g., mean/median values, standard deviation, and beyond) and tools, such as histograms, can be used, depending on the application and types of features.

A remarkable application of statistical methods for data quality regards the **verification of data representativeness**. Some ML model inputs may follow probability distributions (e.g., Gaussian or Weibull). In certain cases, a particular distribution is expected, which can be prescribed by the ML constituent ODD. In this case, one can check whether available data follows such expected distributions by using *goodness-of-fit tests*, such as Chi-Squared test, Kolmogorov-Smirnov test, or Lillie test (the applicability of a particular test depends on the data at hand; sometimes several tests may be applicable and their outcomes can be compared to each other). With such tests it is possible to check whether available data is representative *w.r.t.* the expected distribution (and the ML constituent ODD) or not.

Remark: *Data quality requirements prescribe that datasets must be representative with respect to ML system's operational envelope and ML constituent ODD* [2]. *Even if expected distributions are provided for each of ML model's inputs as part of the ML constituent ODD, certain combinations of inputs may still be invalid. Therefore, rigorous representativeness assessment should consider multivariate distributions to study how inputs move together. Describing and fitting such distributions, as well as goodness-of-fit tests, can be difficult. This poses new challenges for statistical methods, including their scalability and applicability.*

Related EASA AI Concept Paper objectives: DM-06 (data collection), DM-13 (data quality verification).

Comment: Data augmentation via FM is a way to extend the dataset achieving a higher coverage of ML constituent ODD. This is aligned with the expectations of Objective DM-06. Objective DM-13, instead, covers all validation and verification activities for the life cycle data objects pertaining to the data management process, with reference to the data management requirements; this includes evaluation of data completeness, data representativeness, data accuracy, data traceability, datasets independence. As mentioned in this section, data representativeness and completeness criteria can be addressed by adopting statistical methods. For the remaining criteria, Appendix 1 offers an overview of how traditional FM can support the compliance of the data objects with some other data requirements.

4.1.2 Model training

The use of formal methods is also envisioned for improving the training process (learning) by assessing how stable the process is, as well as for guiding it towards high quality models.

FM can be used for **inference of constraints from the data to support training**. Formal properties can be inferred from the data based on certain expected grammar. Such properties can be simple input-output relationships (if-then) discussed in Section 3.4.1.1, arithmetic constraints, and beyond. Such inferred properties can be added as constraints to a customized training algorithm, so that their satisfaction is guaranteed (or violation penalized) during training. This is a possible way of obtaining a trained ML model correct-by-construction with respect to given constraints (see [84], [85], [49]). An emerging area of Physics-Informed Machine Learning (PIML) [86] is an example of using physical constraints in the training process.

The use of formal methods can also be envisioned for **guiding the learning process** to enforce certain properties as constraints during training. This is possible, for example, with employing an iterative

counterexample-guided training process with a verification phase in the loop that attempts to identify property violations and respective counterexamples in the ML model at current training iteration. They can be used to augment the training dataset, thus "navigating" the training towards more satisfiable solutions (*w.r.t.* property of interest) on next iterations. Alternatively, constraints in the objective function of the training (e.g., penalties) can be iteratively tightened to enforce "harder" constraints on a property on next iterations, in case if verification phase concludes that the property is invalid. Some relevant published works include [25] and [49].

Learning algorithm stability addresses modifications to the datasets that are either inherent to the data sources, e.g. noise, or due to deliberate activities. A robust training algorithm should not be significantly affected by the presence of noise in the data. Also, the execution of training on two datasets that are semantically similar or equivalent under certain criteria (e.g., same images under different lighting conditions) should lead to models that exhibit similar behavior. Assessment of the learning algorithm stability can be conducted by modifying or removing one or more points in the training dataset, retraining on this modified dataset and comparing with the result with the model trained on the original dataset. Multiple experiments can be conducted to achieve desired confidence. The possible use of formal methods is envisioned for the selection of points (or regions) to modify in/exclude from the training dataset, based on some distance metric, so that the modified dataset is sufficiently different (or "distant") from the original one. If available, some higher level semantic criterion could be employed (e.g., find and exclude all points with a certain pattern from the training dataset).

Related EASA AI Concept Paper objectives: LM-11 (learning algorithm stability).

Comment: First two applications of this subgroup (inference of constraints from data, guiding the learning process) are not currently mapped to any EASA AI Concept Paper objectives, because they are techniques to obtain a higher quality ML model and do not directly refer to V&V activities recommended by the guidance. Instead, stability verification is directly mapped to the LM-11 objective.

4.1.3 Generalization capability

NOTE: The following is a *special case* and no claim is made that formal methods are generally applicable to solve the generalization assessment problem for ML models. For a more general case, for objectives related to generalization (estimation of generalization bounds) EASA AI Concept Paper suggests as anticipated means of compliance the use of statistical learning theory instruments [2].

Generalization is the capability of an ML model to perform its intended function on previously unseen inputs. In particular cases, when the input space of the ML model is well-defined and bounded[32], formal methods can be employed to **assess the generalization capability of the model**. In such cases it may be possible to split the *entire input space* into subspaces that together cover the input space, and to conduct a formal analysis on each subspace obtaining possible output classes/ranges for *all* possible inputs [87]. Traditional FM techniques, such as abstract interpretation, may be used for this purpose. If there is no input subspace, for which some possible output is wrong and/or exceeds some specified boundaries, then generalizability of the ML model can be concluded with respect to its bounded input space, i.e., a formal proof can be obtained that none of admissible inputs leads to an incorrect or unexpected output.

The approach is illustrated in ***Figure 15***, where the (3-dimensional) input space of the ML model is discretized into smaller "boxes" (hypercubes). Each box has corner points shown in red, and model performance at these

[32] One example of a well-defined bounded input space is a lookup table, which is a typical data structure used as part of many avionics functions. An ML model (e.g., a neural network) can be trained to surrogate such table in order to reduce the memory footprint in the embedded hardware (for example, consider the ACAS Xu collision avoidance system [87]).

points can be checked in a conventional way, i.e., by inference. Formal methods allow to estimate error bounds for each point *inside* the box (exemplified as yellow points), for example, to understand the worst-case error deviation from known corner points. Such deviations from expected values can be used to estimate the difference between ML model errors computed from available data (e.g., test dataset) and "unseen" data points that may occur during operations. Altogether, exhaustive analysis of the boxes leads to full coverage of the input space and, consequently, to a quantifiable generalization guarantee.

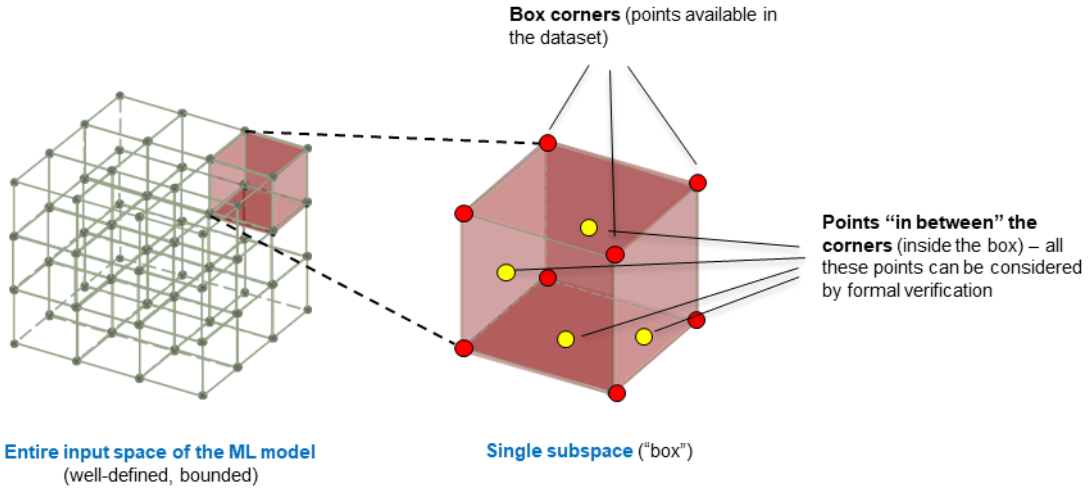

***Figure 15**. Example of well-defined input space split into subspaces to be analyzed by FM.*

Approaches from statistical learning theory (SLT) also provide tools for estimating ML model generalization capability. More precisely, they allow to **estimate the generalization gap** by using metrics, such as VC dimension and others. We refer to the CoDANN-1 IPC report [3] for technical details about SLT techniques.

Related EASA AI Concept Paper objectives: LM-04 (quantifiable generalization guarantees), LM-14 (verification of anticipated generalization bounds).

Comment: Quantification of the ML model generalization guarantees is one of the key assurance objectives (LM-04) prescribed by the existing guidance, while LM-14 intends to validate these guarantees by means of a test (holdout) dataset. FM approaches discussed and referenced in this section are currently limited to low-complexity ML models, such as shallow NNs, while some of them also have constraints on the input space structure and complexity. Statistical learning theory methods may be promising means for analyzing deep learning models, however, their effectiveness has to be better understood [3]. Overall, generalization capability assessment remains an open research problem.

4.2 Formal methods for improving ML model robustness

FM applications from this group contribute to improving the robustness of machine learning models. Potential impact of formal methods spans from verification activities to FM-powered robust training procedures, as well as runtime monitoring of ML models, where unexpected inputs and outputs to/from the ML model could be timely identified and mitigated to avoid unintended behaviors.

Figure 16 illustrates the allocation of FM applications of this group on the W-cycle diagram. Each symbol (square, triangle) refers to one of the subgroups discussed below. Each application is allocated to the W-cycle phase that it can support.

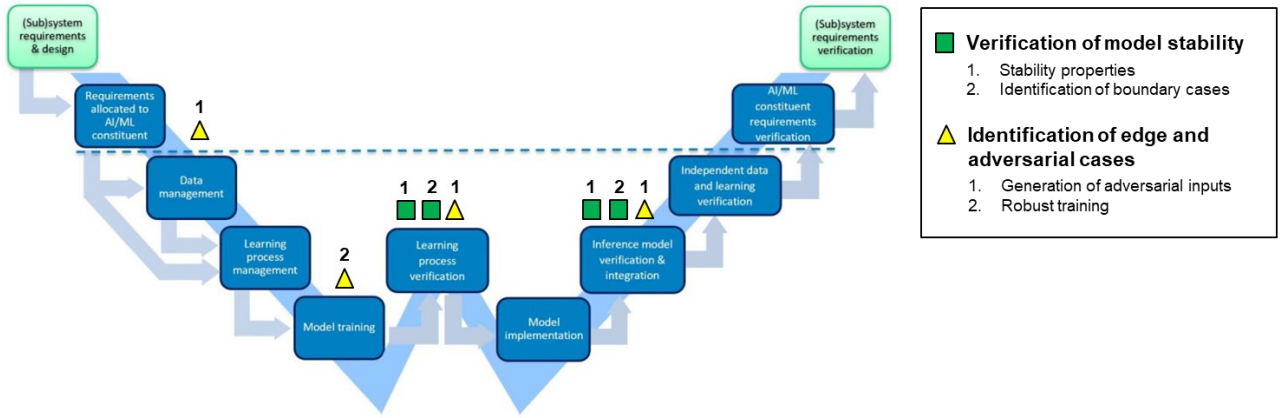

***Figure 16**. Applications of formal methods for improving ML model robustness allocated on the W-cycle diagram.*

4.2.1 Verification of model stability

Verification of stability properties is supported by most of the existing formal verification tools applicable to ML, such as VNN tools (see Section 3.5). Following the definition of stability used in the ForMuLA IPC, the goal of such verification is to check formally defined properties that relate input perturbations to admissible changes in the outputs of the ML model (e.g., class or numeric value). Stability properties consider *normal operating conditions*, i.e., inputs that belong to the ML Constituent ODD[33].

The major part of state-of-the-art FM tools for ML focuses on verifying **local stability** properties, i.e., the effects of input perturbations in the neighborhood of a given point. Local stability properties are usually expressed in the *delta-epsilon* formulation introduced in Equation (3.4), and its variations.

Instead, **global stability property** that is imposed on the entire input space of the ML model, is often out of reach of current FM approaches. This is because the verification problem, as formulated by existing tools, may become intractable if the input space gets large. Some formalizations have still been made available in the literature (e.g., [35]) and, as FM tools mature, such verification may become a possibility, even if tractable only for simpler models, such as shallow NNs.

In general, compliance with ML model stability requirements and objectives requires a global stability property to be valid up to a specified maximum level of input perturbation. Since verification of such global properties is currently impractical, an approximation could be used by replacing a global property with a set of local properties. This approach and the role of the test dataset is further discussed in Section 5.4.2.

Another formal technique to address global stability is the estimation of Lipschitz constants that has been discussed in Section 3.4.1.4. Such FM-based estimators (e.g., [50], [71]) also face the scalability problem for large-scale ML models.

In case of neural networks, if the input space of the ML model is well-defined (e.g., it is hyper-rectangular and can be discretized as point grid), global stability property can be assessed by conducting an exhaustive mathematical analysis over each subspace (e.g., hyper-rectangular cell). In such special cases, growth bounds on NN output value and its gradient can be estimated for each subspace based on the boundary values for that subspace (corner points) that are known. The method has been applied on low-complexity neural

[33] In general, a stability property may still be defined over some input point that belongs to ML constituent ODD (i.e., part of nominal operating conditions), but some of the perturbed inputs generated in the neighborhood of this point may lie *outside* of ML constituent ODD. This may happen, for example, if the chosen input point is close to the boundary of the ODD.

networks [88] but it is unlikely to scale to complex models, such as deep learning. This is also relevant to ML model generalization capability assessment, as previously discussed in Section 4.1.3.

Formal methods can also contribute to **identification of boundary cases**. These are inputs that can be considered "borderline" for the ML model. For example, they may lie close to the decision boundary of a classification model or near a singularity that may cause a regression model to provide outputs that significantly differ from each other. Technically, this can be achieved by solving optimization problems (see Section 3.4.1.3), where the results can be interpreted as boundary cases/points. Methods that are not optimization-based (e.g., abstract interpretation) can also be used. For example, a search procedure can be developed where an FM tool is tasked to check local stability properties with different values of input perturbation until it identifies one that is violated. Boundary cases can then be extracted from counterexamples to this property.

Related EASA AI Concept Paper objectives: LM-12, IMP-07.

Comment: Verification of ML model stability is the subject of objective LM-12, which states: *perturbations in the operational phase due to fluctuations in the data input (e.g., noise on sensors) and having a possible effect on the trained model output*. This includes both nominal cases and boundary cases. While LM-12 regards the trained model, verification of stability is also prescribed by objective IMP-07 for the inference model, where the proposed techniques *may* also be applicable, provided that the tools for reducing SW code to a formal model amenable to analysis by FM technologies discussed in this report are available. Practical demonstration on a use case for LM-12 is provided in Section 5.4.

4.2.2 Identification of edge and adversarial cases

Formal methods can support robustness assessment of ML models by searching for vulnerabilities that, once identified, can be used to develop a robust training procedure and increase the dataset quality. They can also be used for testing purposes. Such information may result either from the specific use of FM tools to search for vulnerabilities, or from counterexamples provided by the tools during the verification of ML properties, such as stability.

One example is the **generation of adversarial inputs** by means of formal methods. For this purpose, some VNN tools that are based on optimization algorithms (e.g., MILP) can be used to mimic an adversary searching for an input to attack the neural network (e.g., see [26], [89], [90], [22]). For example, the delta epsilon property in Equation (3.4) can be reformulated into one of the two optimization problems (see Sec. 3.4.1.3 for details), i.e., *find the smallest delta to exceed the epsilon*, or *find the largest epsilon in the input perturbation space bounded by delta*.

Solutions to such optimization problems are adversarial inputs that, in principle, could be used by an attacker to compromise the NN output[34]. In the assurance process, identifying such inputs can contribute to model robustness. They can be added to the training dataset, so that the model is retrained on an augmented dataset to increase its robustness *w.r.t.* such inputs. A **robust training** procedure that uses FM to search for adversarial examples during (or between) iterations can also be developed [91].

Adversarial examples can also be extracted from counterexamples to property verification problems. For instance, consider a stability property discussed in Section 3.4.1.2. If the property is violated, many FM tools can also return a counterexample, which can be interpreted as a concrete input on which the property does not hold (e.g., a perturbation, that makes the ML model output go beyond admissible error bounds). Similarly

[34] Typical goal of the adversary is to maximize impact of the attack (e.g., damage) and/or not to be detected.

to the above, such counterexamples can be used to complement the training dataset to improve model robustness.

Related EASA AI Concept Paper objectives: LM-13, IMP-08, DM-06.

Comment: Verification of robustness is in the scope of objectives LM-13 and IMP-08 for, respectively, trained model and inference model. Non-requirements based adversarial cases are included in these objectives. Test cases generated by analyzing the ML model using formal methods can be used for testing both the trained model and its implementation, thus contributing to respective objectives. As regards Data Management objectives, proposed FM applications can identify inputs that could complement the training dataset to improve data quality (e.g., data representativeness) and model robustness. Therefore, objective DM-06 that prescribes a data collection process that satisfies data quality requirements becomes relevant, since it mentions synthetic data that can be the output of FM analyses discussed in this section.

4.3 Other formal methods applications for machine learning

This section discusses several other categories of ML-specific FM applications that can contribute to learning assurance, as well as some other objectives of EASA AI Concept Paper. Their suggested allocation on the W-cycle diagram is shown in ***Figure 17***. Each symbol (square, triangle, diamond, hexagon) refers to one of the subgroups discussed below. Note that applications related to explainability (Section 4.3.4) and runtime monitoring (Section 4.3.5) are related to, respectively, AI Explainability and Safety Risk Mitigation building blocks of the EASA Trustworthiness Analysis [2], therefore, they do not directly map to objectives of the Learning Assurance building block. The diagram shows phases of the W-cycle, where these applications can be applied/executed.

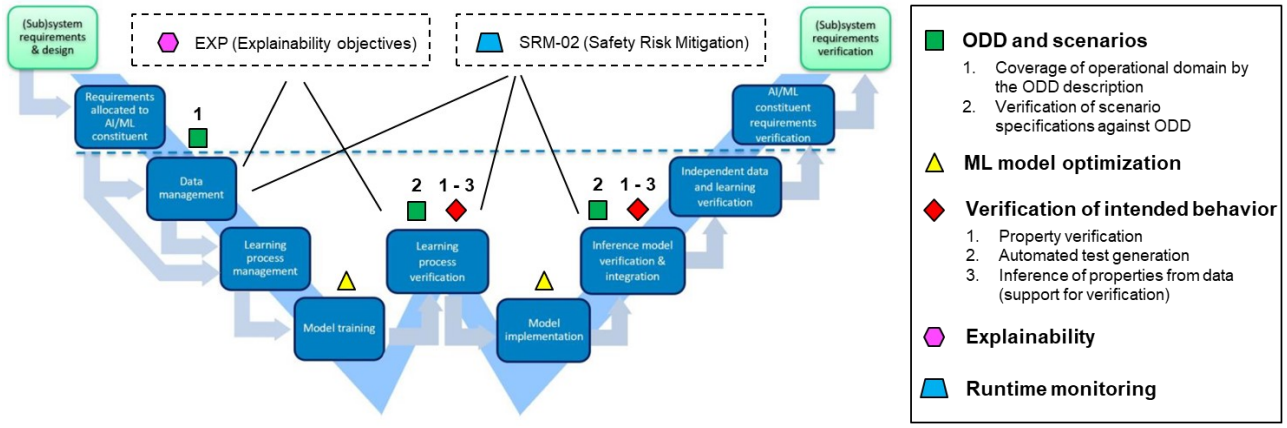

***Figure 17**. Other applications of formal methods allocated on the W-cycle diagram.*

4.3.1 ODD and scenarios

Operational design domain (ODD) defines the range of operating parameters within which the ML-based system is designed to operate as intended (see the full definition in the Glossary). Formalization of ODD[35] is an important step to enable evaluation of ML design artifacts, such as datasets, requirements, and test scenarios, against the ODD to assess their completeness and representativeness. Recent work in the automotive domain led to creation of taxonomies and ontologies for ODD of relevant applications, as well as formal languages for ODD definition, such as the one presented in the ASAM OpenODD concept paper[36]. For

[35] Here ODD is referred to in a broader scope, i.e., at different levels (e.g., ML constituent ODD, system-level ODD). Discussed methods *may* be applied at these different levels.
[36] https://www.asam.net/standards/detail/openodd/

a given ML system/constituent, its defined ODD may not fully represent the *real* operating conditions, called *Operational Domain (OD)* in the same concept paper, which includes all "known" ontology and attributes. Possible role of formal methods is envisioned for **checking the coverage of OD by the elements of concrete ODD**. Here, FM could contribute to identification of gaps in the ODD definition in terms of missing ontology elements that must be considered for the ML system/constituent under design.

Verification and testing of ML models may be challenging due to large input spaces of these models (e.g., in case of vision-based systems), as well as due to the lack of explicitly defined functional requirements for all intended system behaviors. Furthermore, verification would require to observe behaviors of the model in various evolving situations, which are called *scenarios*. A scenario describes the environment (the "scene") and the actions of agents over a period of time. An agent may represent, for example, the ownship vehicle that includes the ML model as part of some ML-enabled system, as well as some other aircraft/entity. Formal languages have been proposed for scenarios, such as ASAM OpenScenario[37]. **Consistency and compatibility of formal scenario descriptions with the ODD description** is another possible application of formal methods relevant to learning assurance.

Related EASA AI Concept Paper objectives: DM-01, LM-10, IMP-09.

Comment: ODD definition is part of objective DM-01, where ODD coverage analysis *w.r.t.* the overall taxonomy of the operational domain may be applicable. Scenario-based testing methods, including formal definition of test scenarios, could be used as part of requirement-based testing activities prescribed both for the trained ML model (LM-10) and the inference model (IMP-09).

4.3.2 ML model optimization

As discussed in Section 3.5.1, FM approaches exist that allow to perform property inference for ML models. In case of neural networks, apart from input-output properties, it is also possible to infer properties that consider inner (hidden) layers of the NN. The latter can constitute a relevant technique for **model distillation** that aims at approximating the behavior of a large neural network with a smaller one that is favorable to deployment under latency and computation constraints [92]. For example, a distilled classification network can only include a subset of layers of the original neural network, given that the properties that map activation patterns of neurons of the last of these layers to the outputs of the original NN have been identified. In such case, remaining layers of the original NN could be replaced with a simple check that maps the activations of the last layer of the distilled NN to an output class. Similarly to neuron pruning, distillation is a model optimization technique.

Related EASA AI Concept Paper objectives: LM-06, IMP-02.

Comment: Model distillation can be applied after the model training is finished. The distillation procedure can be documented – LM-06 – and the distilled model can be validated using FM techniques to guarantee behavior and performance compliance and then used to guide the inference model creation. The implementation of this simplified distilled model can reduce the need of optimizations at the implementation stage, facilitating the adherence with objective IMP-02.

[37] https://www.asam.net/standards/detail/openscenario/v200/

4.3.3 Verification of intended behavior

At the concept level, FM applications specified below are similar to those used for verification in traditional software systems. However, formal specifications of ML models and their properties, such as neural networks, as well as analysis algorithms and tools are new, specific to machine learning domain. That is, traditional solvers (e.g., SMT) cannot be used out-of-the-box to verify intended behaviors of ML models. This motivates to include these novel, yet conceptually traditional, FM applications in this report.

Similarly to verification of stability properties, formal verification tools can be employed to **verify other types of properties** that are formally defined from ML constituent functional and non-functional requirements. For example, the correspondence of certain categories/subsets of inputs to expected output classes (or ranges, in case of regression models) can be formalized as input-output relationships. Properties related to monotonicity of the model (function) output can prescribe, for example, non-decreasing or non-increasing behavior of the output with respect to certain (typically, monotonic) changes in the inputs. Existing FM tools for ML, such as VNN tools, can currently address local (pointwise) monotonicity properties. Formal specifications of ML properties are further discussed in Section 3.4.

Use of counterexamples: As discussed in Section 3.2, traditional FM solvers, such as those based on automated reasoning techniques, are often capable of providing a counterexample in case they detect a property violation (e.g., see [93]). These counterexamples can be further used to improve the quality of the ML model (e.g., stability or monotonicity) by complementing the training dataset and thus serving as extra "examples" for the learning algorithm, as anticipated in Section 4.1.2.

Definition of test cases from functional requirements, as well as robustness testing, are standard methods in traditional software verification. **Automated Test Generation** (ATG) [94] is an FM-based technique to produce test cases in an automatic way with the goal of achieving desired coverage criterion. While structural coverage is often not applicable to learning assurance (e.g., a single test case would typically provide the full coverage of code that implements the neural network), ATG can still be employed to support the coverage of requirements. In the past years, novel techniques have been introduced to perform automated test generation specific to autonomous systems, including machine learning models [95].

Finally, **automated extraction (inference) of properties from datasets** is an innovative recent technique that can extract formal properties, such as logical and arithmetic formulas, from labeled data, based on a given *grammar* (formula templates that are expected to be found). Certain techniques are based on an assumption that all data is correctly labeled [42], while others also allow to tolerate some noise/errors in the labels [43]. Such extracted formulas can contribute to formalization of functional requirements, which, in the case of ML, are often implicitly contained in the data. They can also be used as properties to be later verified on trained and inference models, also by means of formal methods.

Related EASA AI Concept Paper objectives: LM-10 (trained model), IMP-09 (inference model).

Comment: Technologies developed in the recent years are expanding the application of formal methods to new contexts, such as property verification and requirement-based testing for ML models. These novel technologies have an impact on objective LM-10, extending the set of possible means to deal with ML constituent requirements verification and coverage. Traditional FM instead, can be used for requirement-based verification of inference model – IMP-09 – as suggested in Appendix 1.

4.3.4 Explainability

The use of formal methods is also envisioned to support some of the **explainability** objectives in the EASA AI Concept Paper [2]. In particular, FM can be used to produce artifacts for *development explainability*. Such

artifacts could improve developer's understanding of the data and the model under design, facilitate debugging, as well as potentially become a form of a certification evidence.

One example is **inference of properties from datasets**, discussed in Section 3.5.1 and Section 4.3.3. Resulting formal specifications are interpretable and could help to describe the data and understand it, which supports ML model development. Other FM techniques allow to **infer properties from a trained model**, such as neural network, as discussed in Section 3.5.1.1. Such properties (rules) could become useful artifacts for development explainability, since they could improve the knowledge of the ML model behavior by developers. They could be provided to support assurance cases by explaining the rationale behind ML model's decisions in a formally interpretable way. They can also be used in verification, as well as in runtime monitoring.

Related EASA AI Concept Paper objectives: EXP-03, EXP-14 (input monitoring).

Comment: Inferred formal properties could be leveraged as explanations (e.g., "*ML model produces output Y from input X because the property P(X, Y) holds*"), therefore, use of FM could be considered a potential MoC for EXP-03. Additionally, properties inferred from the data can be used to check the validity of inputs during operation, therefore, operational explainability objective EXP-14 that prescribes input monitoring with respect to ML constituent ODD and/or expected data distribution can also be considered. In other words, rules inferred with FM from the data can support out-of-ODD and out-of-distribution detection.

4.3.5 Runtime monitoring

For machine learning, monitoring of the model at runtime (model inputs and/or outputs) is an important **safety risk mitigation** since it may be challenging to guarantee the absence of unintended behaviors over the entire input space of the model. **Runtime monitors** continuously check output values of the inference model and can detect, for example, an unexpected trend in the output, and inform the top-level system or the user that ML model's outputs cannot be considered reliable. Similarly, monitoring of ML model inputs can be carried out to detect out-of-ODD or out-of-distribution inputs. For such inputs, correct outputs of the model may not be guaranteed, and a possible mitigation would be passivate the ML model and switch to a backup function that is proven safe, or simply to notify the user.

Run-time assurance (RTA) architectures add high-assurance components to the system design to ensure that a complex or difficult-to-verify component (such as a ML component) cannot cause unsafe or unintended system behaviors. The ASTM F3269-17 standard for bounded behavior of complex systems [96], also known as a *simplex architecture*, provides guidance for developing RTA architectures that mitigate unintended behaviors through the use of runtime monitors. In this case, the ML component may still produce unintended behavior, but the RTA architecture ensures that this behavior is acceptable in terms of system safety.

One of the steps for design-time assurance is **verifying that the RTA architecture satisfies its high-level requirements**. Traditionally, requirements verification has been achieved using a combination of directed testing methods and manual review. However, model-based specification enables a more rigorous approach to verification via formal methods analysis. With both the requirements and the architecture represented in formal (well-defined, unambiguous) notations, SMT solvers can be employed to determine whether there is any possible sequence of inputs that will violate a requirement. Furthermore, failure by the solver to find a counterexample is essentially equivalent to a mathematical proof the validity of the requirement, i.e., that it can never be violated.

The possible use of formal methods is envisioned for the **automated generation (synthesis) of runtime monitors**. In particular, abstract interpretation methods could be applied to identify *unsafe* input regions. These techniques compute sets of reachable outputs, given an input set (see Section 3.5.2.1). For monitor generation, the analysis can be performed in the opposite direction, i.e., from outputs to inputs (backward reachability). Starting from unsafe outputs, i.e., those that violate some requirement or property, *counter*

input sets (input space regions that lead to these outputs) can be computed by some VNN tools. Clearly, desired neural network performance cannot be guaranteed in such input regions. A monitoring procedure can be employed to check at runtime whether inputs received during operations belong to such *unsafe* input regions. In the latter case, a mitigative action could be taken by the ML constituent, such as a backup function.

An example of such approach is the design of a *hybrid* version of the ACAS Xu airborne collision avoidance controller [87]. Original system design is a set of Look-Up Tables (LUT), which can be approximated and compressed by a set of neural networks, offering a reduced memory footprint [57]. In certain input regions the behavior of the NNs differs from the one of the LUT, which may pose the system at risk. Such *unsafe input regions* have been identified by formal methods, which led to the **construction of a safety net**, which is an extract of the LUT corresponding to these regions. The hybrid controller then combines the neural networks and the safety net through a monitoring mechanism that switches to the safety net in case if an input from an unsafe input region is received during operation.

Another relevant FM-based approach for runtime monitor generation is **inference of properties from datasets**. A number of techniques based on conventional solvers have been proposed (see Section 3.5.1). If these properties are inferred from a reliable dataset, i.e., the one that satisfies data quality requirements, then they can be later used to check new inputs during operations with a runtime monitor. If the input does not violate any property/rule then it may be considered reliable, otherwise it may be discarded or corrected.

Another relevant application is the **automated online construction of a safety envelope** for the ML model. For example, the authors of [25] construct a *monotonic envelope* of a neural network function on-the-fly. Optimization Modulo Theories (OMT) [44] is used to identify counterexamples that maximally violate the monotonicity specification. If the function has a non-monotonic behavior at a given inference point, the value of the upper (or lower) envelope boundary is returned instead of the ML model output in order to preserve monotonicity of the output. Numerical evaluation claims that such online incremental construction of the monotonicity envelope has minor overhead on small models and is, therefore, can be used in applications with small-size neural networks that do not have strict real-time requirements.

Related EASA AI Concept Paper objectives: SRM-02 (Safety risk mitigations).

Comment: Runtime monitoring is in the scope of SRM-02 that states that "*monitoring of the output of the ML constituent and passivation of the AI-based system with recovery through a traditional backup system*" is an acceptable means to be used to *"gain confidence that residual risk is properly mitigated"*. [2] Note that FM applications to generation of runtime monitors, as well as to construction of safety nets/envelopes, may not fall in the direct scope of SRM, because monitors may also be used for other objectives. However, these applications still have an *indirect* impact on SRM, because artifacts generated by FM may be used to detect anbormal inputs/outputs of the ML model or constituent, as well as to make part of a backup algorithm when the ML component is passivated, thus mitigating the residual risk.

5 Assessment of the use of formal methods on the selected use case

The goal of the ForMuLA project is the analysis of the applicability of FM as means of compliance for assurance and certification objectives for ML constituents, and their practical demonstration on a use case. **There is no aim to demonstrate all verification activities**, i.e., to explore and demonstrate all possible FM-based analyses to cover as many data/model requirements as possible. Datasets and models developed and used throughout the project are **work-in-progress** and will be subject to improvements. Some of them have been proposed based on the analyses executed and documented in this section. These improvements are out of scope of ForMuLA, i.e., the project does not intend to develop a perfect RUL estimator. Still, the formal analyses demonstrated in this section are critical to the application and are performed at realistic scale.

5.1 Selection of FM applications to be demonstrated

This section is dedicated to practical demonstration of formal methods on the Remaining useful life use case presented in Section 2. In what follows, a number of FM applications to the V&V of machine learning has been selected for demonstration based on their applicability to the use case, and on the maturity of the method and the tools. ***Figure 18*** illustrates the allocation of these applications on the W-cycle diagram.

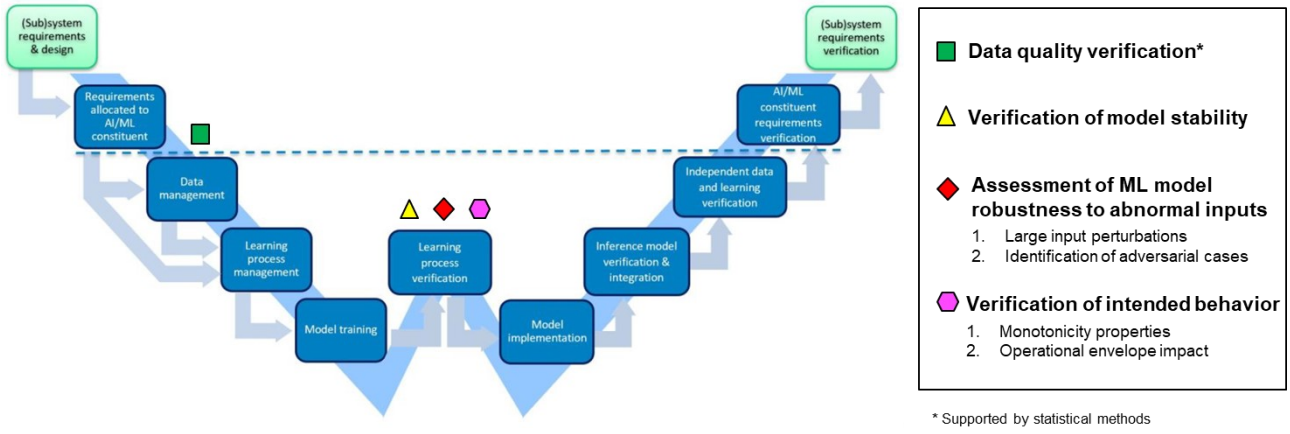

***Figure 18**. FM applications demonstrated in the ForMuLA IPC allocated on the W-cycle diagram.*

Data quality. Data completeness and representativeness are key data quality requirements prescribed by the EASA AI Concept Paper objective DM-13. Representative training data (with respect to ML constituent ODD) contributes to the generalization capability of the ML model. It is similarly important for the test data, because verification results based on representative data provide higher confidence that the ML model meets its requirements. In machine learning, it is often assumed that input data comes from a certain probability distribution (entire data and/or some of its features). For the RUL use case, such distributions are available for certain features as part of the ML constituent ODD. While the use of traditional formal verification methods and tools to analyze probability distributions may be impractical, it is possible to support this analysis with statistical methods. Section 5.3 provides several examples of such approaches.

Verification of trained ML model stability. Stability analysis can be carried out by defining and verifying stability properties on the trained ML model. A number of inputs to the deep learning model for RUL prediction (condition indicators - CIs) are computed from vibration sensor measurements. The sensor can be subject to noise and failures, i.e., perturbations may occur to the CI inputs, and not all of them can be timely noticed. Therefore, it is critical to demonstrate the stability of the RUL estimator to input perturbations. The results of applying a verification approach for stability properties using an abstract interpretation method is demonstrated in Section 5.4.3.

Assessment of trained ML model robustness to adverse inputs. Stability requirements for the ML model prescribe the absence of significant output deviations given a bounded perturbation in the input. Therefore, the inputs for which the perturbation bound is exceeded (i.e., it is greater than δ; see Section 3.4.1.3) can be considered adverse. Robustness of the ML model to such inputs can be verified with formal methods by studying the effects of large input perturbations. For RUL estimator, such perturbations may occur, for example, due to abnormal loads of the mechanical bearing that occur due to some unexpected events (e.g., maneuvers), as well as due to sensor failures. That said, robustness assessment of the trained RUL estimator is a critical assurance activity, fully aligned with EASA AI Concept Paper objective LM-13. Analysis is provided in Section 5.4.4. Additionally, same section demonstrates one of the ways of using FM for identification of adversarial inputs to the ML model.

Verification of intended behavior of the trained ML model. Several other requirements for the trained RUL estimator define its intended behavior, and formal properties can be defined based on them. These properties are amenable to verification with formal methods. Such verification is part of requirements-based testing of the trained model behavior, as prescribed by the EASA AI Concept Paper objective LM-10. Selected properties include monotonicity of the RUL estimator, and the impact of the operating environment on the prediction. Verification approach and results are provided in Section 5.4.5.

5.2 Assessment framework

All selected FM application demonstrators are part of the learning assurance process, as defined by the EASA AI Concept Paper. Their experimental evaluation is supported by an automated toolchain developed by Collins Aerospace Applied Research & Technology. Its key components are discussed in the following sections, with a particular focus on formal verification.

5.2.1 Learning Assurance toolchain

The main intent of the learning assurance toolchain is to enable a model-based process for the development of ML constituents supported by relevant assurance methods at every phase of learning assurance. It currently focuses on the RUL use case and will be further extended to support similar deep learning applications with regression over time series inputs.

The toolchain consists of a set of independent modules. Each of them maps to a phase (or several phases) of learning assurance. Each module accepts specific user inputs (e.g., datasets, requirements) and produces artifacts that are stored in a common workspace, from where they can be accessed by other (subsequent) modules. The main output of the toolchain is the inference model. Each module implements functions, including verification methods, that can be configured and executed by the user via a graphical user interface. Documentation, as well as mapping to objectives of the EASA AI Concept Paper, is also provided in each section of each module. A high-level overview of the learning assurance toolchain is provided in ***Figure 19***.

Each module of the toolchain is briefly discussed below:

1. The **Data management** module is responsible for corresponding data management activities and directly maps to the same phase of the learning assurance process and the W-cycle. It is currently the entry point to the toolchain. Its main functions are data preparation (e.g., feature engineering, normalization, labeling) and data quality verification. Data collection is currently out of scope of this module, therefore, collected raw datasets along with data quality requirements (DQRs) are accepted as module inputs. Various verification methods are available for DQRs. Section 5.3 demonstrates the use of statistical methods for data quality assessment. The main outputs of the module are training, validation and test datasets, as well as some artifacts that summarize the DQR assessment.

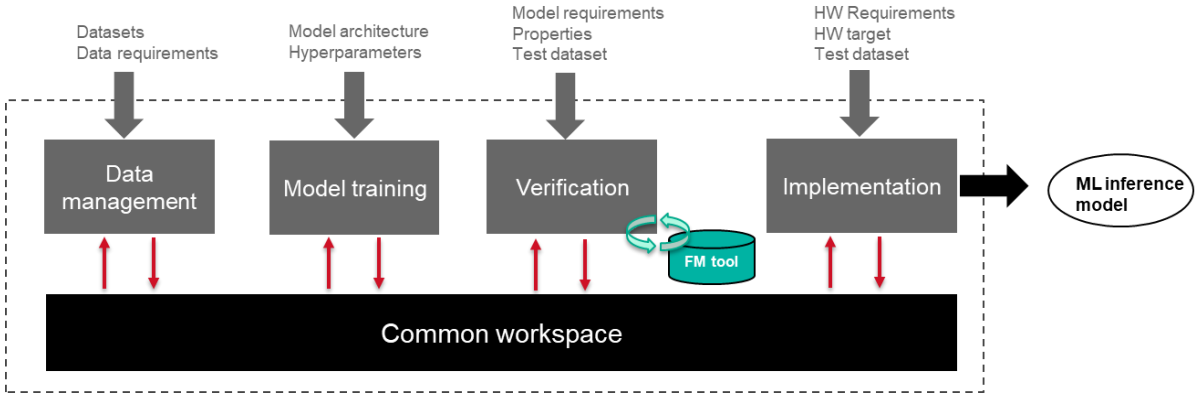

Figure 19. Overview of the learning assurance toolchain.

2. The datasets are then used by the **Model training** module that focuses on learning process management and training of the ML model. User inputs include the model architecture (e.g., NN layers), learning algorithm and objective function selection, as well as hyperparameters (e.g., learning rate, minibatch size, etc.) The module makes use of training and validation sets to train the model, and to optimize its hyperparameters.
3. The key module of the toolchain is the **Verification** module that maps to Learning Process Verification phase in the W-cycle. Along with basic verification functionalities, e.g., performance estimation based on the test dataset, it includes analyses based on formal methods that are demonstrated in Section 5.4. As backend, this module implements a collection of automated property verification procedures that are able to conduct the analyses selected in Section 5.1.
4. Finally, the functionalities of the **Implementation** module are related to creation of the inference model. They include code generation procedures, as well as performance estimation and deployment functions. The output of the module (and of the entire toolchain) is the inference model. This module is out of scope of the ForMuLA IPC and it is not further discussed.

5.2.2 Formal verification framework

The core component of the learning assurance toolchain that is used for the assessment of FM on the RUL use case is the formal verification framework that is part of the Verification module. The framework currently supports formalization and verification of **local properties** to support the analyses discussed in Section 5.1.

5.2.2.1 Property parameters setup

Property formalization is configurable, so that the user can parameterize the properties based on the requirements. More concretely, following parameters can be set:

- **Local stability properties:** input perturbation δ (percentage or absolute value), admissible output deviation ε (percentage or absolute value), where to apply input perturbations, i.e., at which time step, to which features)
- **Monotonicity properties:** percentage γ of input growth rate change, where to apply the growth rate change , i.e., to which features, size of the interval δ for CI growth trajectories.

5.2.2.2 Verification methods

Verification module of the toolchain carries out all property formalization activities, i.e., it creates property objects from numerical values provided in the requirements. It then invokes a VNN tool based on abstract interpretation to verify the properties. The tool computes the output reachable set from 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 "bad" or "unsafe" area). Otherwise, an intersection manifests a property violation. Based on the analyses supported by the VNN tool, Verification module provides the following set of methods:

- **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. The method is *sound and complete* (see definitions in Section 3.2) but may face scalability issues when the complexity of the property and/or the ML model is high.
- **Approximate method.** This method computes over-approximations of possible outputs of all hidden layers of the NN, up to the output layer. This enables faster analysis time and, therefore, scales better than the exact method. The resulting output reachable set is a *conservative approximation* (see Section 3.2), that is guaranteed to fully include the real output reachable set, i.e., the one that would be computed by an exact analysis. If the region associated with property negation (i.e., unsafe region) does not intersect with this over-approximation, it follows that it also does not intersect with the real output set, so that the property can be concluded valid. However, the analysis is *incomplete* and may not be able to disprove the property. If an intersection of the approximated output set an the unsafe region is identified, it could either be a real violation, from which a counterexample can be computed, or a *false negative* (in this case the CEX would be "spurious", i.e., misleading, as discussed in Section 3.2). The tool cannot determine by itself, which of the two is the case. In such situations, to avoid raising a false alarm, the tool returns an "unknown" answer.
- **Simulation-based method.** This is a statistical approach for property *falsification* that is neither sound nor complete. It randomly generates a number of inputs in the neighborhood ("around") the given verification point[38] and performs ML model inference to check the outputs against the property. If a violation is observed, then the property is falsified, i.e., it can be declared invalid. However, the method it is not able to prove the property on a given input set if input values are continuous, because in principle an infinite number of inputs may be generated. Therefore, it is the opposite to the approximate method that can prove the property but may not be able to disprove it. Instead, simulation-based method may identify a counterexample, thus disproving the property, but it can never rigorously prove the property. It is only able to make a *statistical* argument, e.g., use the Monte-Carlo approach to approximate the probability of property violation[39]. One benefit of this approach is that its execution time is constant and does not depend on the complexity of the property that is verified. The method scales linearly with the number of properties/input points.
- **Two-step method.** This approach combines approximate and simulation-based methods. It can be applied when verification with sound and complete methods is impractical. Compared to separately using approximate or simulation-based approaches, it aims at maximizing the thoroughness of the analysis while still keeping solving time at reasonable values[40]. The method is illustrated in ***Figure 20***. 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, simulation-based method is invoked in an attempt to disprove the property by finding a counterexample among a configurable number of randomly generated inputs in the neighborhood of the original point. If no CEX is found, the method returns unknown.

[38] For example, for a local stability property the method generates a number of random perturbations (bounded by δ) in the neighborhood of the given point.
[39] Formally, valid property requires this probability to be 0%, i.e., no counterexamples to be found.
[40] Compared to sound and complete verification; for highly complex models and/or properties the method may also fail to terminate in reasonable time.

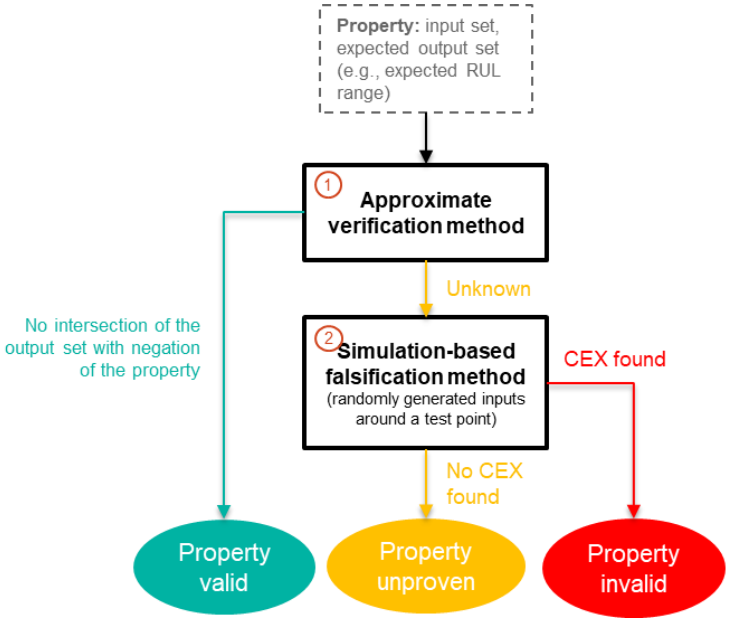

Figure 20. Two-step verification method: approximate reachability (1st step) and simulation (2nd step).

5.2.2.3 Verification process setup

Formal verification framework uses the trained neural network produced by the preceding modules of the learning assurance toolchain. It obtains the NN model from the common workspace in an internal framework-specific format. However, it also accepts models in the ONNX format. This is useful, for example, when the model has been provided by another team, or trained in a different environment.

The type and the number of properties to be verified is also configurable. For local properties, test dataset[41] can be used as a source of input points for which local properties are defined. This leads to the definition of one or more properties for each point in the test set. For example, depending on the requirements, local stability properties can be formulated as perturbations applied to different input features, one or many, which results in multiple properties for the same input point.

The framework also allows to verify specific sets of properties, and has control on the total number of properties to be verified.

5.2.2.4 Verification process results

Following results are collected by the verification framework during its execution:

- For each property
  - Property type
  - Property parameters (e.g., δ and ε for stability properties)
  - Verification result: valid, invalid or unknown
  - Verification time

[41] Quality of the test dataset is an important condition. If the test set meets the data quality requirements, such as completeness and representativeness *w.r.t.* ML constituent ODD, then it is a representative collection of test points. With that, verification of local properties defined for these points may be considered sufficient to provide evidences for learning process verification objectives, such as LM-12, thus increasing the confidence on the correct behavior of the ML model and the absence of unintended functionality. This is further discussed in Section 5.4.2.

- Statistics per property type[42]
  - Number and percentage of valid, invalid and unknown properties
  - Average verification time
  - Total verification time
- Overall statistics
  - Number and percentage of valid, invalid and unknown properties
  - Average verification time
  - Total verification time

***NOTE:** discussion below only aims to provide the "look and feel" of the framework output. Detailed discussion on the verification of monotonicity properties and related requirements can be found in Section 5.4.5.1.*

Intermediate results and final statistics generated by the verification framework are printed to the console, as exemplified in ***Figure 21*** for monotonicity properties verification. First, verification parameters are printed: percentage of CI growth increase within each time window with respect to the original CI trend (delta = 60%) and the type of monotonic shift of inputs ("all CIs", i.e., applied to all condition indicators at once). The total number of related properties is then shown (7493), as well as time elapsed during their initialization. Finally, the verification method is specified (two-step method), and verification result for each property is shown along with elapsed time for its verification. Note that the properties in ***Figure 21*** are proven valid, which is done by the first step of the method, which is approximate method. In this case, simulation-based falsification method is not invoked. Therefore, verification time is shown for the approximate analysis only.

```
Running all CIs with delta 60...
Selected monotonic shift is "all CIs"
Creating monotonicity properties for selected inputs...7493 PROPERTIES
Property creation time: 21.08s
Running verification method    : two-step...
Verifying property (input: 1, feature: all CIs, modification at step: all)...VALID
Verification time (approximate method): 1.79s
Verifying property (input: 2, feature: all CIs, modification at step: all)...VALID
Verification time (approximate method): 0.93s
Verifying property (input: 3, feature: all CIs, modification at step: all)...VALID
```

***Figure 21**. Example console output of the formal verification framework.*

5.2.3 Toolchain implementation

This section describes the concrete tools used to implement the learning assurance toolchain. This information is commercial proprietary and is not available in the public version of the report.

[42] E.g., one can obtain statistics for all stability properties, without considering results for other property types.

5.3 Data quality verification

This section demonstrates the use of statistical methods, discussed in Section 3.7, for the assessment of data completeness and representativeness.

Provided verification examples focus on several critical aspects pertaining to RUL datasets (component degradation sequences), however, **no intent is made to demonstrate the full data analysis** with respect to all relevant criteria and metrics. In the context of the RUL use case, there is a potential of reusing the demonstrated techniques for completeness/representativeness assessment of other features, which are not discussed in this section.

5.3.1 Representativeness of flight regime durations

As discussed in Section 2.2.3, the aircraft can execute several types of flight missions, also called "mission patterns". A number of patterns has been provided by the SME, and they are included in the ML constituent ODD (see ***Table 2*** for an example of a pattern). Each mission pattern is a fixed sequence of flight regimes (e.g., ascent, forward flight, hover), where the duration of each regime follows a probability distribution that is also specified in the ML constituent ODD. A concrete mission is, therefore, a sequence of flight regimes, each of them having a fixed duration that is sampled from a distribution.

Flight regime duration plays an important role in the bearing degradation, because the latter highly depends on the physical load of the bearing. For example, if the aircraft frequently executes missions with relatively long ascent/descent phases, in which the load is higher, then the bearing is likely to wear and degrade faster. In other cases, when flight regimes with smaller component loads prevail, degradation could take more time. To obtain correct predictions, both these possibilities should be considered, i.e., present in the training data. This means that the datasets must be complete, i.e., contain sufficient number of examples, and also be representative *w.r.t.* expected distributions. Hence, the number of regime durations that are close to the distribution mean must be significantly larger than those that are distant from the mean by several standard deviations. However, it is also important to have the latter in the training, as well as in the test dataset.

ML constituent ODD prescribes Gaussian distributions for flight regime durations, with their parameters specified in ***Table 2***. These expected distributions are included into each degradation sequence as metadata and are automatically parsed by the Data Management module of the learning assurance toolchain. Once all sequences are read and preprocessed by the toolchain, representativeness analysis for durations of every flight regime can be conducted by taking following (automated) steps:

1. Create a histogram of durations for the given regime;
2. Perform goodness-of-fit tests with respect to expected distribution (null hypothesis is that the data is accurately described by the distribution at a given level of confidence, the latter is configurable).

A number of goodness-of-fit tests can be applied for the analysis, depending on the data. For example, the **chi-square goodness-of-fit test** determines if a data sample comes from a specified probability distribution. Parameters of this distribution may be either provided by the user or estimated from the data. The test groups the data into a certain number of bins, and then calculates the observed and expected counts of those bins. After that, it computes the following test statistic called the chi-square statistic:

$$\chi^2 = \sum_{i=1}^{N} (O_i - E_i)^2 \Big/ E_i \,, \tag{5.1}$$

where O_i are the observed counts and E_i are the expected counts based on the hypothesized distribution. The test then compares the computed value of the statistic to a chi-square distribution with degrees of freedom

equal to (a) the number of estimated parameters and (b) difference between number of bins for data pooling and the number of parameters, to draw a conclusion on the goodness of fit. Other applicable tests include the Kolmogorov-Smirnov test, the Lillie test, as well as the Anderson-Darling test. Together with the decision on whether the tested sample comes from the specified distribution or not, these tests also provide the p-value based on which the decision was taken.

Results of representativeness verification of flight regime durations with respect to corresponding probability distributions specified in the ML constituent ODD (***Table 2***) is shown in ***Figure 22***. Several regimes are shown: (a) Hover - Short; (b) Hover - Long; (c) ForwardFlight - Short and (d) ForwardFlight – Long. Each diagram includes a histogram showing number of occurrences of regime durations in selected intervals (bins). Horizontal axis has units of time (minutes), left vertical axis corresponds to the number of occurrences for the histogram, while right vertical axis represents the probabilities. Each histogram is overlayed by two probability density functions (PDFs). The solid line PDF shows the expected distribution for the respective flight regime, with parameters prescribed by the ML constituent ODD. Dotted line PDF is specifically fitted to the data using the maximum likelihood method and is shown for comparison purposes (its parameters are estimated from the data). Legend on every plot shows parameters (mean and standard deviation) for both distributions.

Above each diagram, the outcomes of the chi-squared goodness-of-fit tests are shown. The test returns "0" if the null hypothesis is not rejected by the test, i.e., according to the test the data comes from the expected distribution. Instead, if the null hypothesis is rejected at the significance level of 0.05 (5%), then the test returns "1". The choise of the significance level value comes from best practice. For example, for the Hover Short flight regime (***Figure 22a***), the assessment shows values 1/0, which means that the data does not come from the ML constituent ODD distribution ("1"), but it is well described by the fitted distribution ("0")[43]. In other words, the data is not representative of the ML constituent ODD. Same can be observed in ***Figure 22c*** for ForwardFlight (Short). Here, the result does not appear straightforward, because visually the PDF corresponding to the ODD fits the histogram quite well. Still, the fitted PDF is slightly more flattened, and its mean is slightly shifted towards smaller values. Increasing the significance level of the goodness-of-fit test (e.g., to 20%) leads to the null hypothesis not being rejected, however, the justification for such increase has to be further explored. The test instead fails in ***Figure 22b***, because some bins have low expected counts or are empty; the answer is 0/0, but a warning is returned by the framework[44]. This further demonstrates an issue with data completeness, since there are not enough missions in the bearing degradation data that are able to cover well the range of durations for this regime. Finally, the test returns "0" for the ML constituent ODD distribution for regime ForwardFlight (Long), which is shown in ***Figure 22d***. This means that the representativeness test is passed for this regime.

Distributions of flight regime durations can also be analysed more in-depth, i.e., separately for each mission. This analysis has also been performed, however, the dataset size was too small to ensure reliable results of goodness-of-fit tests. Future work shall focus on obtaining larger datasets. In the current report, the feasibility of representativeness assessment via statistical methods has been demonstrated, so that it can be further applied to other features and groups of features[45]. All cases where the actual data had a mismatch with the expected distribution (i.e., was not representative) have been communicated to the SME for further investigation.

[43] Correspondence of the data to the fitted distribution is, of course, expected, but in general not guaranteed (fitting method is independent from the goodness-of-fit test).
[44] The warning message is not visualized in ***Figure 22***. It is generated in the tool console available to the user.
[45] One limitation of this representativeness assessment method is that expected distribution and its parameters, or at least the distribution type, must be known *a priori* and specified in the ML constituent ODD.

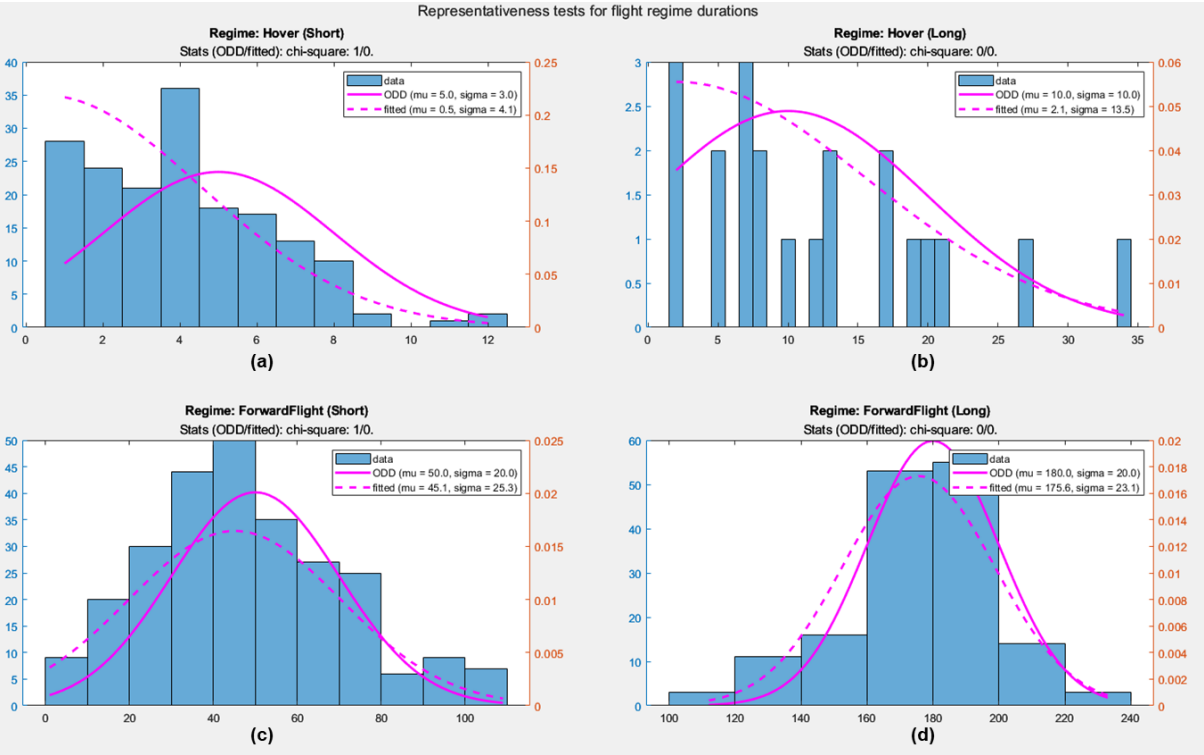

Figure 22. Representativeness assessment results for selected flight regime durations.

Remark: *For the RUL use case, same analysis can also be performed for CIs (condition indicators) because these inputs also have an expected distribution (Weibull) specified in the ML constituent ODD. This analysis is not demonstrated in this report due to time limitations of the project.*

5.3.2 Completeness of mission profiles for different operating environments

Operating environment significantly affects the bearing degradation. Two environments have been considered as part of the ML constituent ODD: desert and normal (non-desert). For the former, small pieces of sand and dust from the atmosphere could penetrate inside the bearing and lead to increased vibrations and, consequently, to faster degradation. Therefore, RUL estimator should be able to distinguish between the two environments, and smaller remaining lifetime should be predicted for the desert conditions.

In each degradation sequence used to train the RUL estimator, the aircraft executes ("flies") a series of different missions, as discussed in Section 2.2.3. Possible mission patterns (sequences of flight regimes with durations described by a distribution) is limited and specified in the ML constituent ODD. To ensure that the ML model is able to correctly account for differences in the operating environment, training data should include "examples" of all types of flight missions in each environment. This is a data completeness requirement. It can be checked by visualizing statistics on how many missions of each type are executed in both environments within the entire training dataset[46]. Result is shown in ***Figure 23***.

[46] Of course, same check must be performed for other datasets (validation, test).

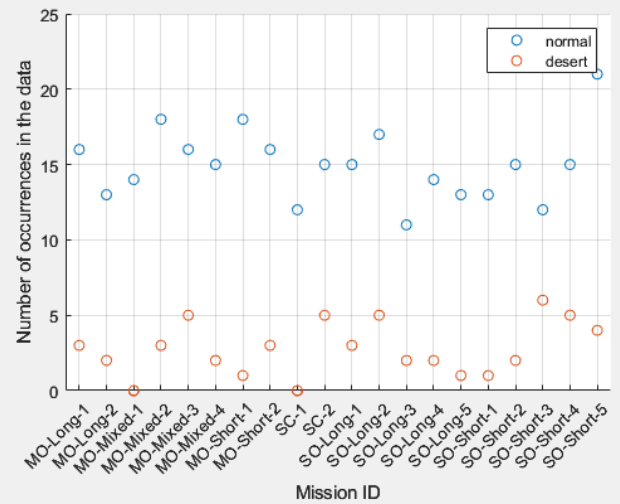

***Figure 23**. Completeness test – coverage of mission profiles by operational environments.*

From the visualization above, data completeness issues are evident. Several flight mission patterns (e.g., MO-Mixed-1, SC-1) do not have any data for the desert environment, which could negatively affect the predictions for these missions. In other words, in the entire available data the aircraft never "flew" any of these two missions in the desert. Also, a *bias* towards the normal environment can be observed. Based on this completeness check, the datasets have been improved to more balanced ones that include sufficient examples for both operating environments for every flight mission.

5.4 Formal verification of the trained ML model

This section provides evaluation results for FM applications selected in Section 5.1 on the RUL use case. First, the details of the RUL estimator that are relevant to the analysis are identified. Then, a detailed description of the V&V activities is given, including the formalization of relevant properties that are traced to requirements in Section 2.2.4.2. For each application, numerical results and discussion are provided.

5.4.1 RUL estimator interface

Recall that the input of the RUL estimator is a time window including the features listed below and of length of 40 time steps (each step equals to 1h). More details are provided in Section 2.2.6. Specifically, following features from Section 2.2.3 are inputs to the estimator function:

1. Seven condition indicators (CIs) that provide numerical information about the bearing degradation status, obtained from the processing of vibration sensor measurements. They are numeric variables of type float, and each of these quantities independently follows a Weibull distribution with different parameter values.
2. Torque, a numeric variable of type float, ranging between 0 and 160.
3. Flight regime, a categorical variable of type string, taking values in {*ground, takeoff, ascent, forward flight, descent, hover, land*}. One-hot encoding is employed to convert the variable into seven binary variables, one per categorical value.
4. Nominal load, a numeric variable of type float, with values in $\{2.1, 9.2, 8.2, 4.0, 7.6, 7.0\}$. Each value corresponds to one of the flight regimes.
5. Environment, a categorical variable of type string, taking values in {*desert, normal*}. Same as the flight regime, it is transformed into two binary variables via one-hot encoding.
6. Mission ID, a categorical variable of type string; a mission ID is associated with a sequence of flight regimes, beginning with *ground* and ending with *land*. Flight regimes have durations independently drawn from a Gaussian distribution with different parameter values.

The only output of the estimator is a numerical value of type integer with non-negative values which represents the remaining useful life of the bearing component in hours.

The knowledge of the domain and of the data characteristics allows to determine a set of relationships and constraints on the inputs and outputs of the ML model:

a. Mission ID uniquely determines a corresponding sequence of flight regimes.
b. Regime, torque, load, CIs, RUL are related as follows:
    a. Flight regime affects the applied torque: it is very low while on ground, low while descending and landing, medium during forward flight, high while hovering, very high during takeoff and ascent.
    b. Flight regime uniquely determines the nominal load value: ground → 2.1; takeoff/land → 9.2; ascent → 8.2; forward flight → 4.0; descent → 7.6; hover → 7.0.
    c. The heavier the applied load, the more stress the bearing is subject to.
    d. Higher torque places heavier load on the bearing, leading to a faster increase of CIs and faster decrease of RUL; however, high torque is not necessarily correlated with a high load.
c. A localized bearing defect, manifesting itself as an increase of the energy value of one CI, tends to have an impact (possible less evident) on the values of other CIs as well; for this reason, a sudden increase of a single CI may denote an issue with the data and should not significantly affect the output of the estimator function. This is captured by requirements RUL-ML-Stab-1 – RUL-ML-Stab-3.
d. In principle, component RUL is a monotonically non-increasing function, and each CI is a monotonically non-decreasing function. However, due to factors such as sensor noise and data inaccuracies, CIs can locally decrease; similarly, a degree of tolerance is to be applied in case of a temporary increase of the RUL. Monotonicity of the RUL estimator is prescribed by requirements RUL-ML-Mon-1 and RUL-ML-Mon-2 (see Section 2.2.4.2).
e. A single degradation sequence corresponds to a scenario, i.e., a sequence of flight missions. The environment is not expected to change throughout the scenario.

5.4.2 Use of test dataset

Non-functional requirements of the RUL estimator are imposed on the ML constituent in its entire ODD. That is, formal properties that are defined for selected requirements (e.g., stability, monotonicity) are in fact global properties. Formal guarantees on the validity of these properties are desirable to demonstrate that the requirements are met. For that, the properties must be shown to hold on *any* input to the RUL estimator that is within its ODD. However, due to the high complexity and the dimensionality of the ML constituent ODD, formal analysis of all possible inputs belonging to the ODD is intractable. Existing VNN tools are currently not scalable to such analysis.

As verification of global properties cannot be achieved, one needs to rely on approximating this analysis. A reasonable approximation would be to reduce the analysis to a set of input points and verify local properties for these points. The result of such replacement of global property verification with a set of local pointwise verifications could be acceptable if altogether selected input points and corresponding properties are representative of the ML constituent ODD and cover it sufficiently. In other words, a discretization of the ML constituent 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 the data quality requirements prescribed by the objective DM-13 of the EASA AI Concept Paper, each dataset (training, validation, test) must be complete and representative with respect to the ML constituent ODD. Relevant data verification techniques, such as the one demonstrated in Section 5.3, could

be applied to assess these characteristics of the data. Since the verification of both trained and inference ML models has to be performed on a **test (holdout) dataset**, as prescribed by the existing guidance, as well as by the common ML practice, this dataset could be used as an approximation of the ML constituent ODD, provided that its completeness and representativeness are demonstrated[47].

Test dataset for the RUL estimator is obtained by (i) concatenating degradation sequences in an arbitrary order and (ii) flattening this concatenation into a numbered list of time windows. That is, the inputs are *locally* adjacent and consecutive, but 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 consecutive). There is no specific ordering of degradation sequences in the test dataset, they are independent from each other. Overall, the concatenated sequences result in 7493 time windows that are inputs to the CNN model. Test dataset contains 15% of overall available data for the RUL estimator (remaining 70% and 15% are allocated to, respectively, the training set and the validation set).

Completeness of the test dataset has been assessed with several methods to make sure it contains sufficient "examples" from different ODD regions. In particular, completeness and representativeness analyses shown in Section 5.3 have been employed. Revealed issues with dataset completeness have been mitigated by adding additional degradation sequences that complement identified gaps, while some of the identified problems with representativeness of certain features are currently under investigation.

Remark: *From previous paragraph, it follows that in the ForMuLA IPC no claim is made that available data is complete and representative. The purpose of the project is to demonstrate **how** certain analyses (including data quality) can be carried out using formal methods and statistical methods.*

5.4.3 Stability verification of the trained ML model

5.4.3.1 Formalization of stability properties

Following notation is used in the remainder of the report. Bold font (e.g., $\boldsymbol{x}$) is used to denote a vector or a matrix, depending on the context, while normal font (e.g., x) denotes a scalar.

Recall that a single input point[48] for the RUL estimator function corresponds to an $L \times n$ time window of predefined number of time steps L with n features ($1 \leq i \leq n$). Formally, an input point is a matrix $\boldsymbol{x} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_n]$, 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 $\boldsymbol{x}_i = [x_i^1, \dots, x_i^L]^T$.

Assumption. *In the following, when considering two input points $\boldsymbol{x}$ and $\boldsymbol{x}'$, 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 $j \notin S, t \neq t^*$.*

Trained model stability verification considers perturbations of certain inputs to the RUL estimator, namely the CIs, which are anticipated to be up to **40%** of their initial values, as per requirement RUL-ML-Stab-1, while an acceptable discrepancy between actual and expected outputs is up to **10h**, as per requirements RUL-ML-Stab-2 and RUL-ML-Stab-3 (see Section 2.2.4.2). Stability properties can be formulated either with respect to a single CI, or a subset of CIs, assuming that other inputs do not change; applied perturbation is limited to a single step in the time window. The rationale of this setting is that, on one hand, a resonance frequency being excited could temporarily increase one or multiple CI values. On the other hand, the persistence of such perturbation could be interpreted by HUMS data quality indicators as an issue in the sensor measurements,

[47] Note that completeness and representativeness of the dataset are considered **necessary** conditions for approximating a global property with local properties defined for the elements of this dataset.
[48] In the following, "input window" and "input point" are used interchangeably, both meaning a single element in the test dataset.

so that the data would be flagged as invalid and not provided to the ML Constituent, i.e., no RUL prediction will be computed (see Section 2.2.3.2 on ODD monitoring). Therefore, perturbations to single or multiple CIs over multiple steps in the time window are not considered realistic, and corresponding properties are neither formulated nor verified.

To overcome the limitations of FM in dealing with the verification of global properties of deep neural networks and to provide evidence for aforementioned stability requirements, local stability properties have been formulated and verified for each point in the test dataset, as motivated by Section 5.4.2.

Stability to 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 local stability property for a single CI is:

$$\forall \boldsymbol{x}' : \left| x_i'^t - x_i^t \right| < \delta |x_i^*| \Rightarrow |f(\boldsymbol{x}') - f(\boldsymbol{x})| < \varepsilon \tag{5.2}$$

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 perturbed value of the CI i at time step t; $f(\boldsymbol{x}')$ is the ML model output computed from the perturbed input), x_i^* represents the average initial value of the CI i, computed over all degradation scenarios, 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 parametric formulation of the stability property is

$$\forall \boldsymbol{x}' : \forall i \in S : \left| x_i'^t - x_i^t \right| < \delta_i |x_i^*| \Rightarrow |f(\boldsymbol{x}') - f(\boldsymbol{x})| < \varepsilon \tag{5.3}$$

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

Considering stability requirements RUL-ML-Stab-2 and RUL-ML-Stab-3 with $\delta = 40\%$ (same perturbation magnitude for each CI) and $\varepsilon = 10h$ for a given input, concrete formulations of local stability properties for, respectively, single CI perturbation and multiple CI perturbations become

$$\forall \boldsymbol{x}' : |x_i'^t - x_i^t| < 0.4 |x_i^*| \Rightarrow |f(\boldsymbol{x}') - f(\boldsymbol{x})| < 10h$$

$$\forall \boldsymbol{x}' : \forall i \in S : \left| x_i'^t - x_i^t \right| < 0.4 |x_i^*| \Rightarrow |f(\boldsymbol{x}') - f(\boldsymbol{x})| < 10h$$

Informally, if the value(s) of the CI does (do) not change at a step t of the analyzed time window by more than 40% of their initial value(s), then the RUL predicted for that time window must not deviate by more than 10h at t.

5.4.3.2 Verification process summary

Properties. Two types of stability properties have been considered: stability to a single CI perturbation (verified for each of the CIs separately), and stability to simultaneous perturbations to all CIs, as described in the previous subsection.

Datasets. The analysis was executed on a test dataset organized as described in Section 5.4.2.

Time steps. CI perturbations were applied to time step 20 (out of 40) of each time window; a single step per window was considered sufficient for the analysis, due to time windows being adjacent. Although the analysis encompassed the entire degradation scenario, emphasis was given to the last **100h** of the RUL, identified as a *critical range* in the requirement RUL-ML-14 (Section 2.2.4.1, ***Table 5***).

Verification methods. Both the exact method and the two-step method described in Section 5.2.2.2 were chosen for evaluation. These approaches are complementary. On one hand, the exact method is sound and complete; it guarantees to return a correct answer (valid/invalid) for any property, but in practice its execution can be infeasible due to time and/or memory considerations. The two-step method, on the other hand, is itself the combination of more efficient incomplete techniques, i.e., approximate verification and simulation. Approximate verification is sound in determining the validity of a property, but not complete: it may possibly return a false alarm, as discussed in Section 3.2. Therefore, instead of *invalid* answers this method reports *unknown* answers. Simulation is used to compute the actual output for a given set of input points, and can be used to check (again, without guarantee of completeness) whether the unknowns obtained from approximate verification actually correspond to invalid properties. This is a property falsification approach, which is detailed in Section 3.7.

5.4.3.3 Verification results

Verification process had 3 phases. All results are summarized below.

- **Phase 1.** Exact verification method was tested in the presence of a single CI perturbation at a single time step, as formulated in Equation (5.2). The encoding produced ~52500 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.
- **Phase 2.** Exact method was tested on properties involving simultaneous perturbations of all CIs at a single time step, according to Equation (5.3). The encoding produced ~7500 properties, one per input time window. The method could not verify any property within a time limit of **3h,** or sometimes also due to failures caused by out-of-memory events. **This is a scalability limitation.**
- **Phase 3.** Two-step verification method was tested on the same properties as in Phase 2. The adoption of an approximate method allowed to overcome the difficulties experienced with the exact method: all properties have been 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.

Results are summarized in ***Table 10***, 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) – total, as well as average times for valid, invalid, and unknown properties (in brackets). Total verification time is also provided in the last column. It can be noted that Phase 2 is redundant to Phase 3 and in fact provided no results, however, it is also shown here to demonstrate the scalability problem. Since in Phase 3 invalid and unknown (unproven) properties have been present, the table additionally specifies the results for the critical range of the RUL to see whether some of these properties could have potential safety impact.

***Table 10**. Verification results for local stability properties of the RUL estimator.*

setting	#properties	#valid	#invalid	#unknown	avgTime (s)	totTime (s)
Phase 1 (single CI, exact method)	52451	52451	0	0	0.3255 (0.3255, -, -)	17074
Phase 2 (all CIs, exact method)	7493	-	-	-	Timeout (-, -, -)	Timeout
Phase 3 (all Cis, 2-step method)	7493	7369	6	118	0.6347 (0.57, 6.49, 4.18)	4756
	1414 **(critical range)**	1414	0	0	0.3313	468.45

Figure 24 shows how 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[49] local stability properties correspond to adjacent and consecutive time windows. Black solid vertical bars in ***Figure 24*** mark the ends of the degradation sequences, i.e., 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.

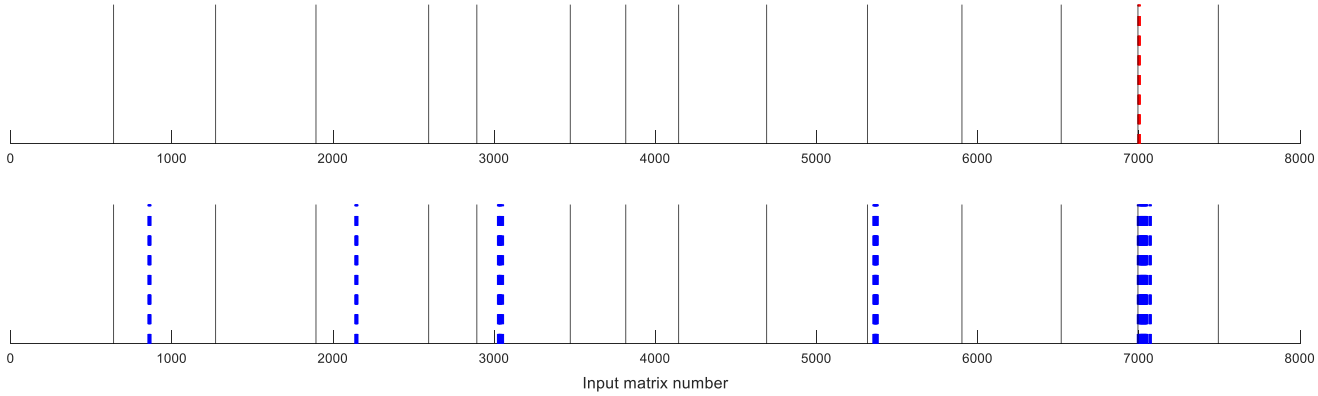

Figure 24. *Location of inputs related to invalid (top) and unknown (bottom) properties for all sequences in the test dataset.*

According to verification results, unknown and invalid properties have been identified in several degradation sequences, but they tend to be concentrated within the first half of the sequence time frame. Focusing on the only degradation sequence where invalid properties appear (***Figure 25***), it becomes clear that the RUL estimator behavior is problematic at the very beginning of the sequence. There are only 6 invalid properties, which are preceded and followed by a larger number of unknown properties. These properties may also be invalid, but the simulation-based part of the two-step verification method was not able to falsify them, i.e., to identify any counterexample.

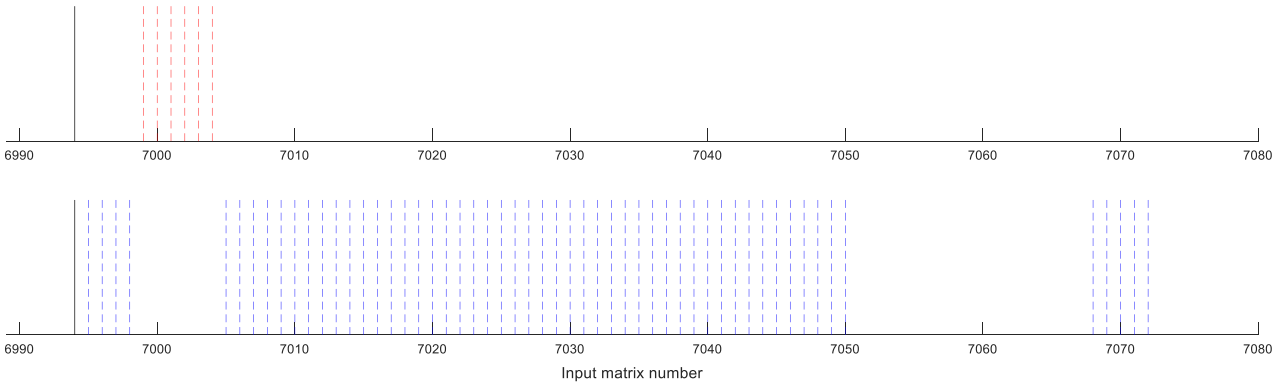

Figure 25. *Location of inputs related to invalid (top) and unknown (bottom) properties in a selected degradation sequence.*

[49] Recall that test dataset is obtained as a concatenation of all time windows in all degradation sequences, i.e., as a flattened list of time windows (see Section 5.4.2 for details). Degradation sequences are independent from each other. Sequence boundaries are shown on the plots to understand where violated/unknown properties are located relatively to their degradation sequence.

Noteworthy, when limiting the Phase 3 analysis to the critical range (last 100h of remaining lifetime of the bearing) for each degradation sequence in the test dataset, all properties pertaining to this range are valid. This increases confidence in the correctness of the behavior of the RUL estimator with respect to safety considerations discussed in Section 2.2.7. Consider the last row of ***Table 10***. Accuracy and stability of the estimator in the critical range are particularly important, because of the possible safety impact related to the incorrect RUL estimation (e.g., resulting from a perturbed input).

Verification results are further discussed in Section 5.5.

5.4.3.4 Visualization of valid properties

Verification framework allows to visualize the information about valid properties. An example is shown in ***Figure 26*** that corresponds to one of the verified stability properties with multiple CI perturbations. The sub-charts show CI trends over the input window (in blue color), except the last one that shows (again, in blue) the RUL value for that time window. Black horizontal bars denote the perturbation bounds for each CI ($\pm\delta$), while on the last subchart they denote the admissible RUL deviation ($\pm\varepsilon$). Several different perturbations with different magnitudes $\delta \leq 40\%$ have been applied to each CI (green circles). It can be seen that the estimated RUL never exceeded $\varepsilon = 10h$.

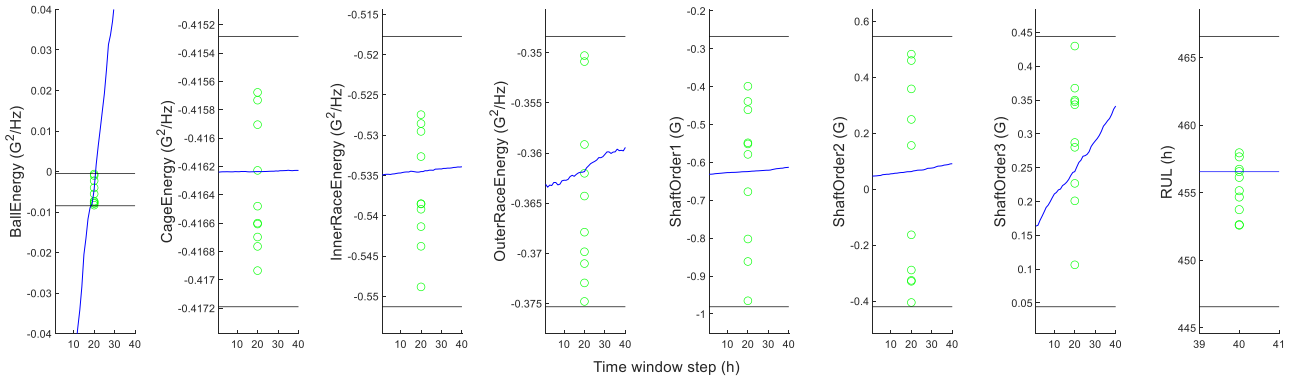

Figure 26*. Visualization of a valid local stability property.*

5.4.3.5 Visualization of counterexamples

For each property that has been proven invalid by the verification framework, one or more counterexamples have been provided. For stability properties, all counterexamples come from the simulation-based method, i.e., they are the result of applying bounded random perturbations on selected CIs corresponding to the property. This is because exact method was only applied to verify stability properties with single CI perturbations, and all these properties have been proven valid. It did not scale to properties with multiple perturbed CIs, where invalid properties have been identified only by the two-step method.

It is important to analyze counterexamples to gain insights on the possible reasons of property violation. Verification framework permits visual analysis of the counterexamples. An example is illustrated in ***Figure 27*** for a stability property to multiple (all) CI perturbations at a single time step.

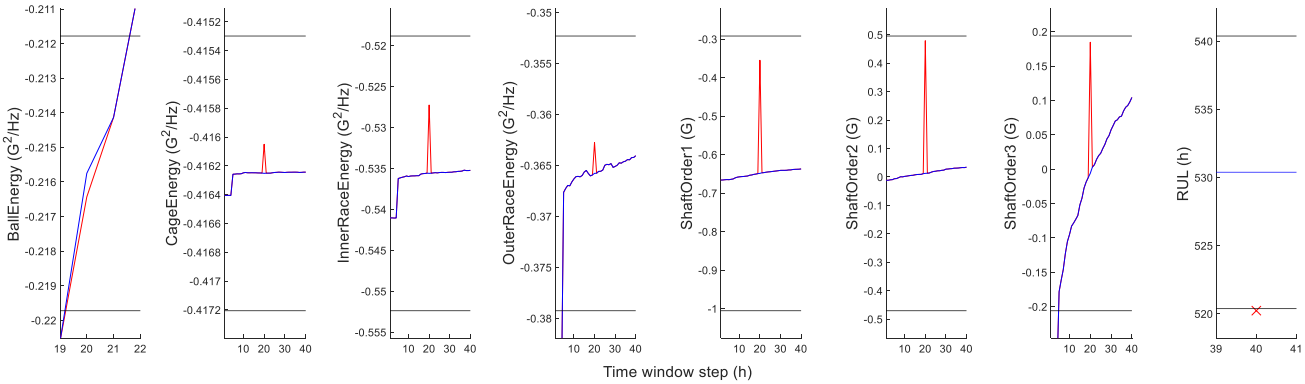

Figure 27. Visualization of a counterexample to a local stability property.

First seven sub-charts 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 $\varepsilon = 10h$.

Several observations can be made:

1. Each CI received a positive perturbation, i.e., a spike towards higher values that are associated with higher bearing degradation.
2. Three last 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 ***Figure 27***. Therefore, such counterexample demonstrates a non-stable behavior of the RUL estimator, i.e., **stability requirement RUL-ML-Stab-3 is not currently met**.

5.4.4 Robustness assessment of the trained ML model

5.4.4.1 Summary of the analysis

Robustness analysis intends to assess the performance of the RUL estimator in the presence of unexpected deviations. This is particularly relevant to the CI inputs. Possible reasons for CI perturbations to occur with unexpectedly high magnitudes vary from abnormal aircraft maneuvers that may lead to a spike in the bearing load, to unexpected environment conditions. Requirements RUL-ML-Stab-2 and RUL-ML-Stab-3 prescribe a stable behavior of the ML model in the presence of input CI perturbations bounded by δ. Formal analysis has been conducted to determine the effect of CI perturbations that exceed this boundary, thus stepping outside of the admissible perturbation range. The analysis boils down to verification of stability properties formulated same as in Section 5.4.3.1 (perturbations on multiple CIs), but with an increasing value of δ. The results for each selected δ are compared to observe the effect of increased perturbation magnitudes on the RUL, i.e., to study how the model performance degrades in the presence of high (more than expected) input perturbations.

5.4.4.2 Verification results

Robustness assessment has been executed by verifying local stability properties for values of $\delta \geq 40\%$ for CI perturbations, which exceeds the maximum admissible CI perturbation prescribed by requirement RUL-ML-

Stab-1. Such perturbed inputs are unlikely to occur and can be considered out-of-distribution inputs. Testing of the ML model on such inputs is prescribed by the objective LM-13 of the EASA AI Concept Paper [2].

A number of perturbation magnitudes with increasing δ (step of 10%) has been explored – up to 70%, which was considered sufficient by the SME. Results are summarized in ***Table 11***. Trends for the number of valid, unknown and invalid properties, as well as the verification time, are shown in ***Figure 28***. Same as in ***Table 10***, verification time reports the average time for all outcomes, and, separately (in brackets), average times for valid, invalid, and unknown properties.

Table 11*. Results of robustness assessment of the RUL estimator.*

setting	range	δ	#properties	#valid	#invalid	#unknown	avgTime (s)	totTime (s)
2-step method $\boldsymbol{\varepsilon = 10h}$	full	40%	7493	7369	6	118	0.6347 (0.57, 6.49, 4.18)	4756
		50%	7493	7119	146	228	1.1267 (0.89, 6.29, 3.04)	8442
		60%	7493	6528	528	437	1.8961 (1.26, 7.31, 4.89)	14208
		70%	7493	5462	1285	745	3.11 (1.70, 8.53, 4.09)	23303
	critical	40%	1414	1414	0	0	0.3313	468
		50%	1414	1414	0	0	0.6078	859
		60%	1414	1414	0	0	0.9969	1409
		70%	1414	1414	0	0	1.4954	2114

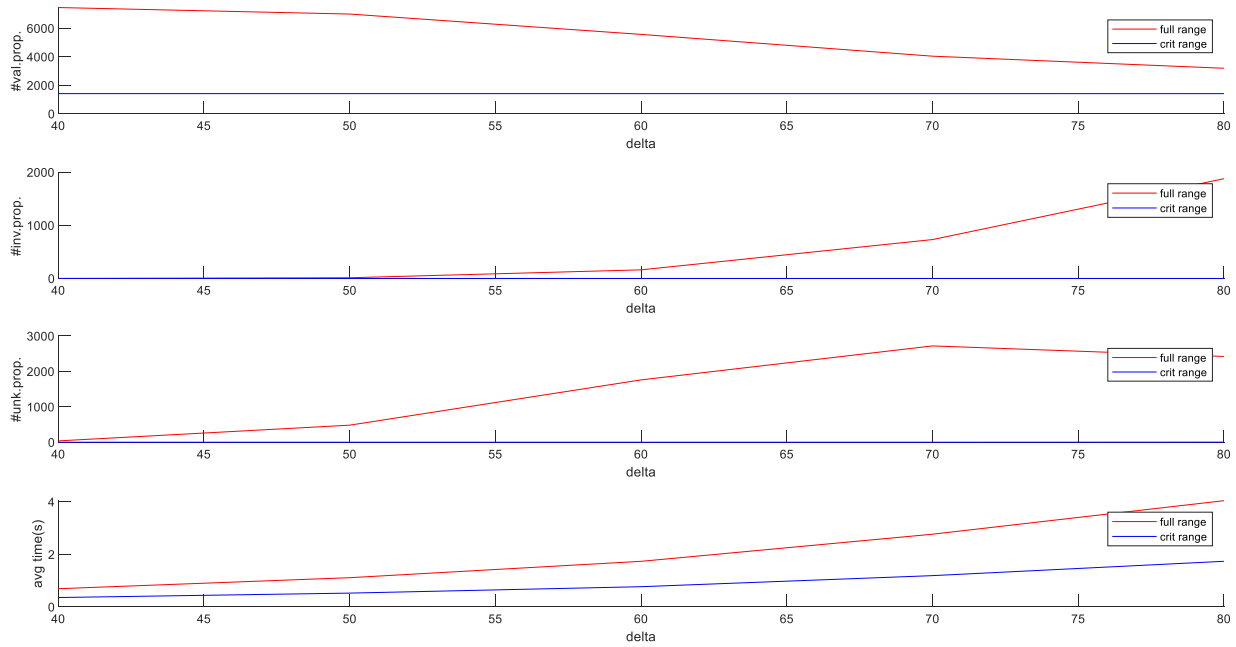

Figure 28*. ML model robustness assessment for increasing CI perturbation magnitudes (over 40%).*

The impact of the perturbation magnitude δ on the results is noticeable: as δ grows one can observe an increase of unknown and invalid properties. This is because with the larger perturbation applied to CI, resulting input to the RUL estimator becomes more "distant" from the original test point, therefore, it becomes more likely that this difference will be reflected in the RUL value. Noteworthy, increasing

perturbation magnitudes **did not affect the critical zone**[50] of RUL estimator, i.e., degraded component states in which its RUL is below **100h** (no invalid properties have been identified for perturbations of up to 70%). This is an important result that demonstrates the robustness of the ML model at component states where possible safety impact may occur.

Identified counterexamples are very similar to the one shown in ***Figure 27*** with the difference that some of the CIs are perturbed by a value (δ) that exceeds 40%. These CEX are omitted for brevity.

From the computational perspective, the growth of the verification time is evident. This is because, as δ grows, the solver has to consider a larger input space for each property, and this increases (often, significantly) the analysis time. Perturbation magnitude is one of the key aspects of stability property complexity that affects the scalability of existing FM tools. It can also be observed that detecting the invalidity of a property is more computationally demanding than proving its validity. This is because the two-step method is used. If the property is valid, only the first step (approximate reachability analysis) would suffice to prove it by showing the absence of intersection between the property negation and the over-approximated output set of the ML model. If an intersection is found, the second step (running simulations) is invoked in an attempt to falsify the property by identifying a counterexample. This brings additional overhead for verification.

5.4.4.3 Identification of adversarial cases

An adversarial case relates to an input that causes significant performance degradation of the ML constituent, with a low probability of being detected during input preprocessing or runtime monitoring. In other words, such inputs, often called “adversarial attacks”, intend either to maximize performance degradation or to minimize the input modification while still exceeding some performance degradation boundary (e.g., change of an output class). As discussed in Section 4.2.2, formal methods can be employed to search for adversarial inputs in a form of solving an optimization problem (maximize output deviation or minimize input perturbation while significantly changing the output). VNN tools based on optimization are applicable for such tasks.

In the current verification framework, FM backend is based on abstract interpretation and thus does not support such optimization problem solving. However, smallest input change that results in a property violation could be identified by setting up a procedure that *iteratively* verifies same properties while varying their parameters related to ML model input change. This approach is exemplified on local stability properties with the goal of identifying a smallest perturbation δ for condition indicators that leads to a violation of some property. Differently put, this is a form of sensitivity analysis of the RUL estimator to the magnitude of CI fluctuations that can determine a threshold value of the δ parameter at which unknown and/or invalid properties begin to be reported by the verifier.

A simple technique to identify such threshold consists in iteratively performing stability verification according to a binary search on the interval $[0\%, 40\%]$. The upper bound of 40% comes from the requirement RUL-ML-Stab-1. It is the highest admissible devitation for each input CI, therefore, to remain undetected, adverse perturbations should stay below 40%. Analysis focused on identifying the smallest δ for which perturbations on all CIs at a single time step lead to RUL estimation that exceeds $\boldsymbol{\varepsilon = 10h}$ (violation) or to impossibility of proving the property (unknown) for some input points. Results are reported in ***Table 12***.

[50] This is an *empirical* observation from the verification conducted in this project. Additional analysis of the data and/or the ML model would be required to explain the result (e.g., using more test data pertaining to the critical range, cross-validation, etc.).

Table 12. Identification of a boundary CI perturbation for the RUL estimator.

setting	range	#prop	#valid	#invalid	#unknown	avgTime(s)	totTime(s)
$\delta = 20\%$	full	7493	7493	0	0	0.1805	1352.51
$\delta = 30\%$			7488	0	5	0.3693	2767.24
$\delta = 25\%$			7493	0	0	0.2803	2100.12
$\delta = 27.5\%$			7493	0	0	0.3263	2445.12
$\delta = 28.75\%$			7493	0	0	0.3536	2649.44
$\delta = 29.375\%$			7491	0	2	0.3639	2726.84
$\delta = 29.0625\%$			7493	0	0	0.3522	2639.63
$\delta = 29.21875\%$			7492	0	1	0.3553	2667.84

Smallest input perturbations that may lead to ML model instability are around $\delta = 30\%$. More precisely, the very first unknowns have been detected at a value slightly larger than 29% (last row in ***Table 12***). These perturbation magnitudes are smaller than those prescribed by the stability requirement RUL-ML-Stab-1, where $\delta = 40\%$. For the current model they can be considered smallest input changes to obtain an adversarial result (in this case, hampering ML model stability).

5.4.5 Verification of intended behavior of the trained ML model

Several RUL estimator functional requirements that define its intended behavior can be formalized as properties and then verified on the trained ML model. This section reports the verification approach and the results for these requirements.

5.4.5.1 Monotonicity properties

Differently from stability requirements, the requirements on monotonicity of the RUL estimator concern all steps of the input window, i.e., within the entire 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). Monotonicity requirements prescribe a non-increasing behavior of the estimator given a change in the growth rate (higher or smaller) of one or more condition indicators.

Condition indicators are correlated with component degradation and failures. Their values are expected to *monotonically increase* during the use of the bearing component, which reflects its degradation. Consequently, the bearing RUL is expected to *monotonically decrease*. Expected behavior of the RUL estimator output is monotonic with respect to the inputs (CIs), i.e., when CIs increase the RUL should decrease. Monotonicity requirements in ***Table 6*** consider realistic situations when the bearing degradation rate increases (for example, due to some failure that develops in the bearing). Faster degradation means that CIs (one or many) grow faster. Expected behavior of the RUL is monotonic, i.e., it should decrease faster, or at least not increase.

Formalization of requirements RUL-ML-Mon-1 and RUL-ML-Mon-2 leads to global monotonicity properties, same as for the stability analysis in Section 5.4.3. These properties would be challenging to address with formal verification. A reasonable approximation to such global properties, as discussed in Section 5.4.2, would be a set of *local* properties specified for input points from a representative test dataset. In the following, the approach and the results of such analysis of local monotonicity properties is presented.

According to the definition of monotonicity properties provided in Section 3.4.1.5, a monotonic[51] change in selected input features of the ML model, all other features being constant, shall lead to a monotonic change in the output. For the RUL estimator, the increase of one or more condition indicators manifests component degradation and, consequently, the decrease of the RUL. A shift in the CI over the time window may be expressed as adding a constant term to the CI value at each time step, as illustrated in ***Figure 29a*** for ShaftOrder3 CI (blue line – original CI trend, red line – increased by a step change at every time step). However, such step changes over the entire window are rather unexpected and will not be considered. Instead, a realistic CI increase, as prescribed by RUL-ML-Mon-1 and RUL-ML-Mon-2, is an increase of its growth rate within the time window. ***Figure 29b*** illustrates such increase, where the blue line is the original CI trend in the time window, and the red line (with higher slope) is the CI with increased growth rate.

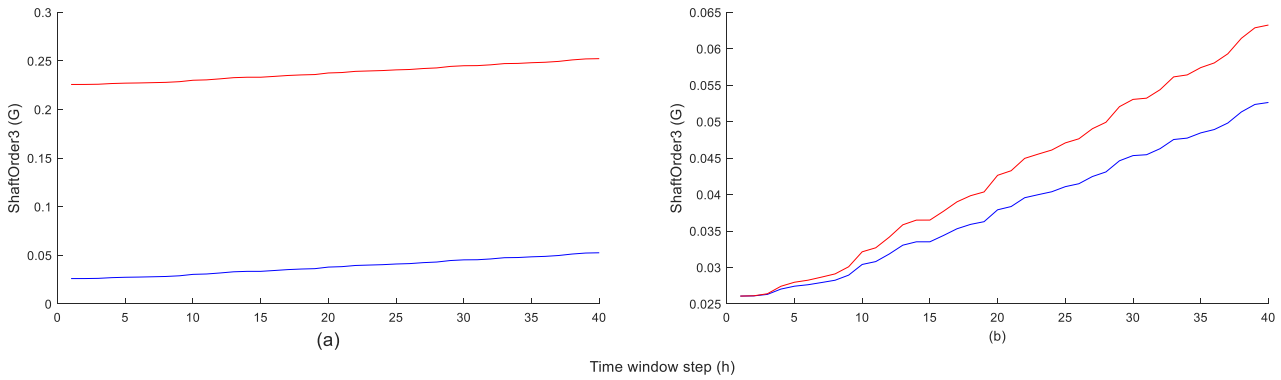

Figure 29*. (a) A step change in the CI over the input time window; (b) Growth rate change of a CI.*

Similarly to stability verification, to overcome the limitations of existing FM tools for verifying global properties, the problem of verifying global monotonicity properties has been reduced to the verification of a set of local monotonicity properties. The latter describe possible changes of CI growth rates in the neighborhood of a representative set of points, i.e, the test dataset (see Section 5.4.2). Formalization of such local monotonicity properties has required multiple phases (1-4) to deal with scalability, coverage, and effectiveness issues. Each phase considered properties with the change in (1) single CI and (2) all CIs simultaneously, as prescribed by the requirements.

Phase 1. A growth rate increase of a CI by some percentage (as exemplified in ***Figure 30a***) represents a different CI "trajectory" within the time window. Such new trajectory can be used as an upper bound, while the original CI growth trend represents a lower bound. The strategy is to analyze all possible CI trajectories within these bounds, eventually reducing the verification of a global property to verification of such intervals specified by multiple local properties. 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. Local monotonicity property for this CI can be formulated as

$$\forall \boldsymbol{x}' : \forall t : x_i^t \le x_i'^t \le x_i^t + \gamma \left| x_i^t - x_i^1 \right| \Rightarrow f(\boldsymbol{x}') \le f(\boldsymbol{x}), \tag{5.4}$$

where prime ($'$) denotes a *modified* item (i.e., $\boldsymbol{x}'$ is the time window, where one or more CIs have modified growth rates *w.r.t.* the original time window $\boldsymbol{x}$; $x_i'^t$ is a modified value of the CI i at time step t; $f(\boldsymbol{x}')$ is the ML model output computed from the modified input), and γ (gamma) is the parameter that regulates the CI slope change[52]. This property states that for *any* growing CI trajectory bounded by the original CI trajectory

[51] Note that the term "monotonicity" applies both to the notion of the function (RUL estimator) and of the input sequence (the values of features in the time window).

[52] Informally, γ can be interpreted as the angle by which the CI slope is increased or decreased with respect to the original one. Note that no particular value of γ has been prescribed in the monotonicity requirements. Therefore, this study evaluates a set of representative values corresponding to realistic CI slope changes.

(lower bound) and the one changed by a percentage γ (upper bound), i.e., a steeper growth trend, the RUL shall be non-increasing *w.r.t.* the original CI trajectory. The difference $\left|x_i^t - x_i^1\right|$ represents an approximation of the CI slope in the interval $[1, t]$.

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. Consequently, also the overall intervals to be analyzed by formal verification are substantially large, even if the upper bound (increased CI slope) is small. With that, based on experience with stability properties in Section 5.4.3, exact verification method would not be applicable. Therefore, the two-step method with approximate verification via abstract interpretation has been selected. However, several input windows have been evaluated, and no result was provided within a timeout of 3h, even for a growth rate change of a single CI. Therefore, **scalability did not permit to proceed with the analysis**, and other approaches have been identified, as described below.

Phase 2. To address the scalability problem, instead of verifying all possible growing CI trajectories within an interval, the behavior of the RUL estimator has been evaluated for a set of steeper CI slopes (with respect to original rates) for each test dataset point, with γ varying from 10% to 50%. Variation of γ has been chosen according to SME feedbacks. Corresponding property for a single CI i is formalized as follows:

$$\forall \boldsymbol{x}' : \forall t : x_i'^t = x_i^t + \gamma\left|x_i^t - x_i^1\right| \Rightarrow f(\boldsymbol{x}') \leq f(\boldsymbol{x}) \tag{5.5}$$

Informally, for a given input, if the growth rate of the CI value increases by a percentage of the original rate then the output RUL should not increase. Similarly to Equation (5.4), the difference $\left|x_i^t - x_i^1\right|$ represents an approximation of the CI slope in the interval $[1, t]$. Formalization for multiple CI slope changes is similar. Illustration of different CI slopes is provided in ***Figure 30b*** with both higher and smaller CI growth rates *w.r.t.* the ones in the original time window**.** The analysis does not consider intervals of possible CI growth trajectories, but only single trajectories obtained from the original one with a given slope change (γ). Therefore, simulation-based methods can be used to obtain the results.

Verification results for single CI slope changes (cumulative results for each distinct CI) and simultaneous slope change for all CIs are shown in, respectively, ***Table 13*** and ***Table 14***. For the single CI case, corresponding to requirement RUL-ML-Mon-1, the number of invalid properties is on the order of 20%, gradually reducing to 15% with the increasing (steeper) slope of the CI. Instead, increasing simultaneous growth of all CIs causes the estimator to correctly identify the decrease in the RUL, with only few (around 0.3%) properties being invalid. Noteworthy, violations also occur in the critical range of RUL, i.e., high degradation states (in case of of simultaneous growth of all CIs in ***Table 14***, *all* invalid properties belong to the critical range). The tables also report average time per property and total time for the verification of the entire test dataset.

Phase 3. The main drawback of the simulation-based verification conducted in Phase 2 is the ***coverage*** of the input space. For each test point only a selected small number (one per each γ value) is verified, even though various CI growth trajectories may occur "in between" the increased slope and the original one. At the same time, the desirable analysis (Phase 1) faces scalability issues. Therefore, the formulation of local monotonicity properties has been updated to address the coverage problem. The intent is to keep the verification time reasonable while still verifying a large number of possible CI trajectories, rather than single trajectories as in Phase 2.

Current analysis phase still considers a number of increasing CI slopes for selected values of γ, however, it defines a space of possible CI trajectories in the neighborhood of the shifted CI (see ***Figure 30c***):

$$\forall \boldsymbol{x}' : \forall t : x_i^t + \gamma\left|x_i^t - x_i^1\right| - \delta \leq x_i'^t \leq x_i^t + \gamma\left|x_i^t - x_i^1\right| + \delta \Rightarrow f(\boldsymbol{x}') \leq f(\boldsymbol{x}) \tag{5.6a}$$

$$\forall \boldsymbol{x}' : \forall t : x_i^t - \gamma\left|x_i^t - x_i^1\right| + \delta \geq x_i'^t \geq x_i^t - \gamma\left|x_i^t - x_i^1\right| - \delta \Rightarrow f(\boldsymbol{x}') \geq f(\boldsymbol{x}) \tag{5.6b}$$

where prime ($'$) denotes a *modified* item (i.e., $\boldsymbol{x}'$ is the time window, where one or more CIs have modified growth rates *w.r.t.* the original time window $\boldsymbol{x}$; $x_i'^t$ is a modified value of the CI i at time step t; $f(\boldsymbol{x}')$ is the

ML model output computed from the modified input). Parameter γ regulates the CI growth rate change (same as in Equation 5.4), and δ is a constant that controls the width of the space of possible CI trajectories. The difference $\left|x_i^t - x_i^1\right|$ represents an approximation of the CI slope in the interval $[1, t]$. Equation (5.6a) captures the property for steeper CI slopes, i.e., having higher growth rate compared to the original CI slope in the time window, and require a non-increasing RUL. Similarly, Equation (5.6b) considers shallower CI slopes and prescribes a non-decreasing RUL in that case.

__Table 13__. Monotonicity property verification results (Phase 2, single CI, increased CI growth rates).

setting	range	gamma	#prop	#valid	#invalid	avgTime(s)	totTime(s)
Sim-based method **single CI, steeper slope**	full	10%	52451	41928	10523	0.0250	1313.94
		20%		43029	9422	0.0259	1357.32
		30%		43591	8860	0.0254	1331.59
		40%		43929	8522	0.0254	1335.04
		50%		44163	8288	0.0270	1415.28
	critical	10%	9898	8096	1802	0.0252	249.35
		20%		8267	1631	0.0265	262.17
		30%		8335	1563	0.0254	251.34
		40%		8378	1520	0.0259	256.20
		50%		8398	1500	0.0262	259.36

__Table 14__. Monotonicity property verification results (Phase 2, all CIs, increased CI growth rates).

setting	range	gamma	#prop	#valid	#invalid	avgTime(s)	totTime(s)
Sim-based method **all 7 CIs, steeper slope**	full	10%	7493	7469	24	0.0283	211.77
		20%		7468	25	0.0270	204.99
		30%		7468	25	0.0272	203.89
		40%		7468	25	0.0271	202.93
		50%		7468	25	0.0271	202.96
	critical	10%	1414	1390	24	0.0286	40.52
		20%		1389	25	0.0273	38.66
		30%		1389	25	0.0271	38.34
		40%		1389	25	0.0268	37.93
		50%		1389	25	0.0268	37.96

For simultaneous growth rate change of all CIs, as in requirement RUL-ML-Mon-2, the properties are formulated in a similar way and are omitted for brevity.

Two-step verification method provided answers for each property, because the space of CI trajectories to be analyzed is smaller *w.r.t.* Phase 1. All results are, however, either unknown or invalid. This is because the formulation does not prevent "oscillating" CI trajectories, i.e., highly non-monotonic ones, since it only defines lower and upper bounds for CI values at each time step, but not prescribes any interdependency between the consecutive values. See example in ***Figure 31***. Such CI trajectories may represent excessive noise but in general are not realistic. At the same time, they invalidate every property, **hampering the effectiveness of the verification**. Hence, the analysis space has to be further restricted.

Phase 4. One of the major challenges with monotonicity properties formalization and verification was the characterization of a realistic sequence of CI values. Although it is expected that CI values globally exhibit an increasing trend in any time window, local oscillations due to noise in the data or other factors prevent the sequences from being monotonically increasing. A compromise solution has been found as a tradeoff

between input space coverage, analysis scalability, and adherence to the ML constituent ODD. It consists in extending the CI shift 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 30d***).

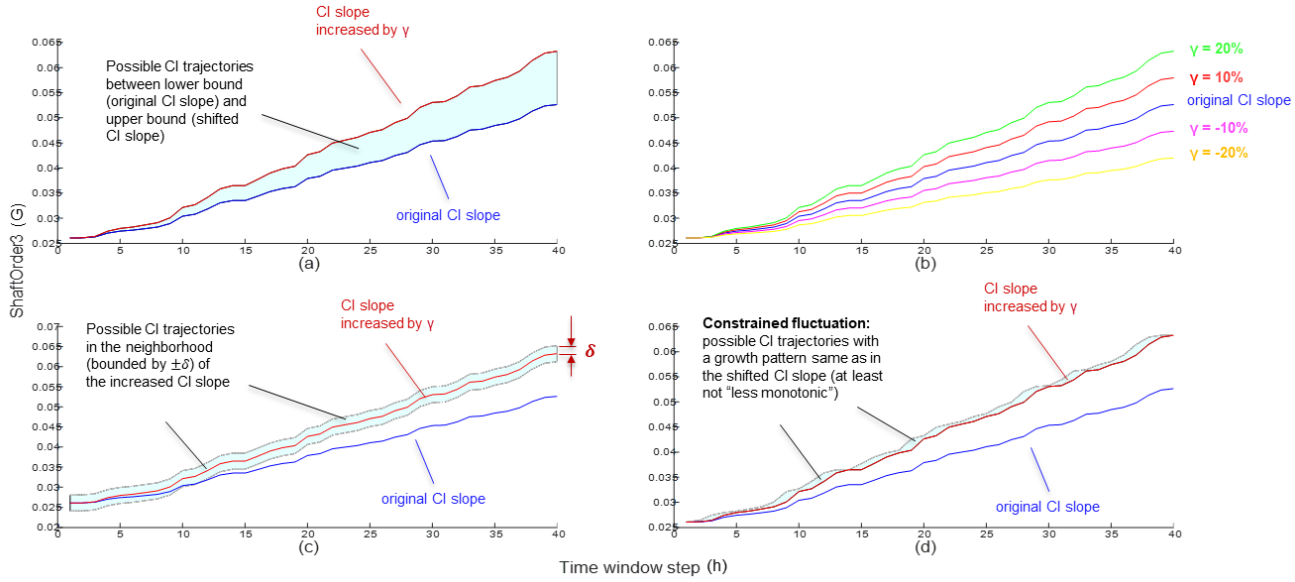

***Figure 30**. Phases of monotonicity analysis.*

(a) Entire space of possible CI trajectories between original and shifted trajectories (Phase 1); (b) Set of CI trajectories with larger and smaller growth rate w.r.t. *original trajectory (Phase 2); (c) Subspace of CI trajectories defined in the neighborhood of the shifted CI trajectory, bounded by $\pm\delta$; (d) Subspace of CI trajectories with constrained fluctuation (Phase 4).*

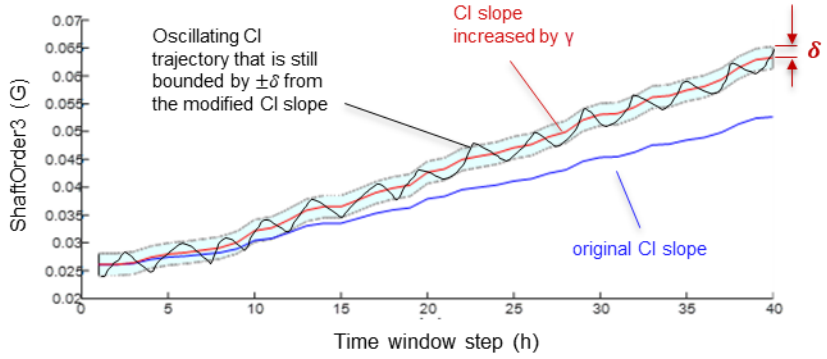

***Figure 31**. Example of an oscillating CI trajectory that invalidates the properties in Phase 3.*

Given a CI i, step t, constant γ, let $u_i^t = x_i^t + \gamma|x_i^t - x_i^1|$ and $v_i^t = x_i^t - \gamma|x_i^t - x_i^1|$. Local monotonicity properties with constrained fluctuation for a single CI growth rate change can be expressed as:

$$\forall \boldsymbol{x}' : \forall t : u_i^t \le x_i'^t \le \max(u_i^t, u_i^{t+1}) \Rightarrow f(\boldsymbol{x}') \le f(\boldsymbol{x}) \quad (5.7a)$$

$$\forall \boldsymbol{x}' : \forall t : \max(v_i^t, v_i^{t+1}) \ge x_i'^t \ge v_i^t \Rightarrow f(\boldsymbol{x}') \ge f(\boldsymbol{x}) \quad (5.7b)$$

Consider the Equation (5.7a). 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 $u_i^{t+1} \ge u_i^t$, otherwise it is set to u_i^t, as in Equation (5.5). 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 $x_i^t \le x_i^{t+1}$,

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 (5.7b) 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.

The formulas are generalized to the case of shifts applied to multiple CIs, as follows:

$$\forall \boldsymbol{x}' : \forall i \in S : \forall t : u_i^t \leq x_i'^t \leq \max(u_i^t, u_i^{t+1}) \Rightarrow f(\boldsymbol{x}') \leq f(\boldsymbol{x}) \quad (5.8a)$$

$$\forall \boldsymbol{x}' : \forall i \in S : \forall t : \max(v_i^t, v_i^{t+1}) \geq x_i'^t \geq v_i^t \Rightarrow f(\boldsymbol{x}') \geq f(\boldsymbol{x}) \quad (5.8b)$$

where S is a subset of indexes of input features corresponding to CIs.

Analysis has been performed both for each single CI slope change (and corresponding constrained fluctuation interval of CI trajectories) and for all CI slopes changed simultaneously. Both *steeper* and *shallower*[53] slopes, i.e., higher and lower CI growth rates *w.r.t.* the original CI, have been considered, with γ varying from 10% to 50%. Same as in Phase 3, two-step method has been used. Results are shown in ***Table 15*** - ***Table 18***. Tables report 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.

Table 15*. Monotonicity property verification results (Phase 4, single CI, increased CI growth rates).*

setting	range	gamma	#prop	#valid	#invalid	#unknown	avgTime(s)	totTime(s)
2-step method **single CI, steeper slope**	full	10%	52451	41572	7797	3082	0.1716	8999.18
		20%		42875	7624	1952	0.1660	8706.01
		30%		43513	7568	1370	0.1591	8346.08
		40%		43869	7543	1039	0.1632	8563.28
		50%		44108	7524	819	0.1608	8436.01
	critical	10%	9898	8061	1455	382	0.1620	1603.92
		20%		8248	1446	204	0.1568	1551.75
		30%		8316	1445	137	0.1498	1482.74
		40%		8359	1434	105	0.1533	1517.10
		50%		8387	1426	85	0.1515	1499.57

Table 16. *Monotonicity property verification results (Phase 4, all CIs, increased CI growth rates).*

setting	range	gamma	#prop	#valid	#invalid	#unknown	avgTime(s)	totTime(s)
2-step method **all 7 CIs, steeper slope**	full	10%	7493	7364	59	70	1.049	7858.03
		20%		7458	26	9	1.075	8052.77
		30%		7457	25	11	1.068	8005.68
		40%		7458	24	11	1.104	8274.89
		50%		7458	22	13	1.080	8098.44
	critical	10%	1414	1373	26	15	0.984	1392.05
		20%		1379	26	9	1.010	1428.61
		30%		1378	25	11	1.004	1419.15
		40%		1379	24	11	1.027	1451.68
		50%		1379	22	13	1.008	1426.42

[53] Mechanical degradation of bearing components suggests increasing CI trend as the only one expected, therefore, higher CI growth rates (steeper slopes) must be considered to assess monotonicity properties. In this study, lower CI rates (shallower slopes) have also been considered to evaluate the *sensitivity* of the RUL estimator to different growth rates.

Table 17. Monotonicity property verification results (Phase 4, single CI, decreased CI growth rates).

setting	range	gamma	#prop	#valid	#invalid	#unknown	avgTime(s)	totTime(s)
2-step method **single CI, shallower slope**	full	10%	52451	36450	13984	2017	0.1792	9399.18
		20%		41508	10114	829	0.1666	8740.32
		30%		43242	8726	483	0.1621	8500.42
		40%		43966	8163	322	0.1588	8327.44
		50%		44288	7965	198	0.1259	8230.24
	critical	10%	9898	6877	2454	567	0.1722	1705.08
		20%		7965	1755	178	0.1560	1543.98
		30%		8218	1570	110	0.1537	1521.87
		40%		8324	1488	86	0.1504	1488.42
		50%		8382	1451	65	0.1483	1467.76

Table 18. Monotonicity property verification results (Phase 4, all CIs, decreased CI growth rates).

setting	range	gamma	#prop	#valid	#invalid	#unknown	avgTime(s)	totTime(s)
2-step method **all 7 CIs, shallower slope**	full	10%	7493	6254	887	352	1.089	8158.06
		20%		6975	291	227	1.057	7923.25
		30%		7274	117	102	1.068	8000.55
		40%		7376	71	46	1.168	8755.21
		50%		7401	55	37	1.079	8087.06
	critical	10%	1414	1180	162	72	1.032	1459.96
		20%		1334	52	28	0.985	1393.47
		30%		1380	13	21	0.992	1403.40
		40%		1401	1	12	1.081	1528.29
		50%		1405	0	9	1.006	1422.03

Given the results summarized in ***Tables 15 - 18***, the following observations can be made:

- The estimator is more likely to correctly react, i.e., exhibit monotonic behavior, to simultaneous change in all CIs rather than a single CI. Considering increased CI growth rates for single CI and all CIs (resp. ***Table 15*** and ***Table 16***), 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. To better understand why individual CI growth rates cause a higher number of property violations, additional investigations have been conducted. The largest number of violations occur with condition indicators related to the shaft (ShaftOrder1 and ShaftOrder3) – at least a half of these properties is invalid. Shaft is a cross-component for the bearing, so if the shaft health is decreased substantially (which is manifested by increased shaft CI growth rate), this could have a multiplicative effect on the RUL of the bearing. In other words, the estimator may indeed be more sensitive to changes in shaft CIs. A large (around 30%) amount of unknown answers have also been provided for the CageEnergy CI growth rate changes, which requires further investigation.
- The estimator is more monotonic when CI growth rate change is large: $\gamma = 50\%$ has much fewer property violations compared to $\gamma = 10\%$. 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 the RUL.

- For decreased CI growth rates (shallower slopes), same trend can be observed: bigger changes in γ lead to the estimator being more monotonic. While it is unexpected for a CI to start decreasing during operation, a noteworthy observation from this analysis is that the estimator's behavior seems more random for shallower slopes. This can be seen, for example, from ***Table 16*** and ***Table 18***: possible CI trajectories around the increased CI slope by $\gamma = 10\%$ result in a total of 129 invalid and unknown properties, while for decreased CI slope by same percentage, the number of invalids and unknowns amounts to 1239, which is an order of magnitude difference. This possibly suggests that the estimator's sensitivity to changes grows as the slope diminishes.

Statistics on property validity and average verification time (in seconds) per property for Phase 4 experiments are shown in ***Figure 32***. The trend discussed above can be observed: the behavior of the estimator becomes more monotonic for higher changes in the CI growth rate, both for individual CIs and multiple (all) CIs. 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. The verification of properties that consider multiple CIs 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 by 7 CIs) must be considered for all CI properties. Therefore, the input space is larger for the latter analysis. Overall, total verification times for both types of properties are comparable, since more properties need to be verified for individual CIs (each CI is considered separately), even if average verification time per property is smaller.

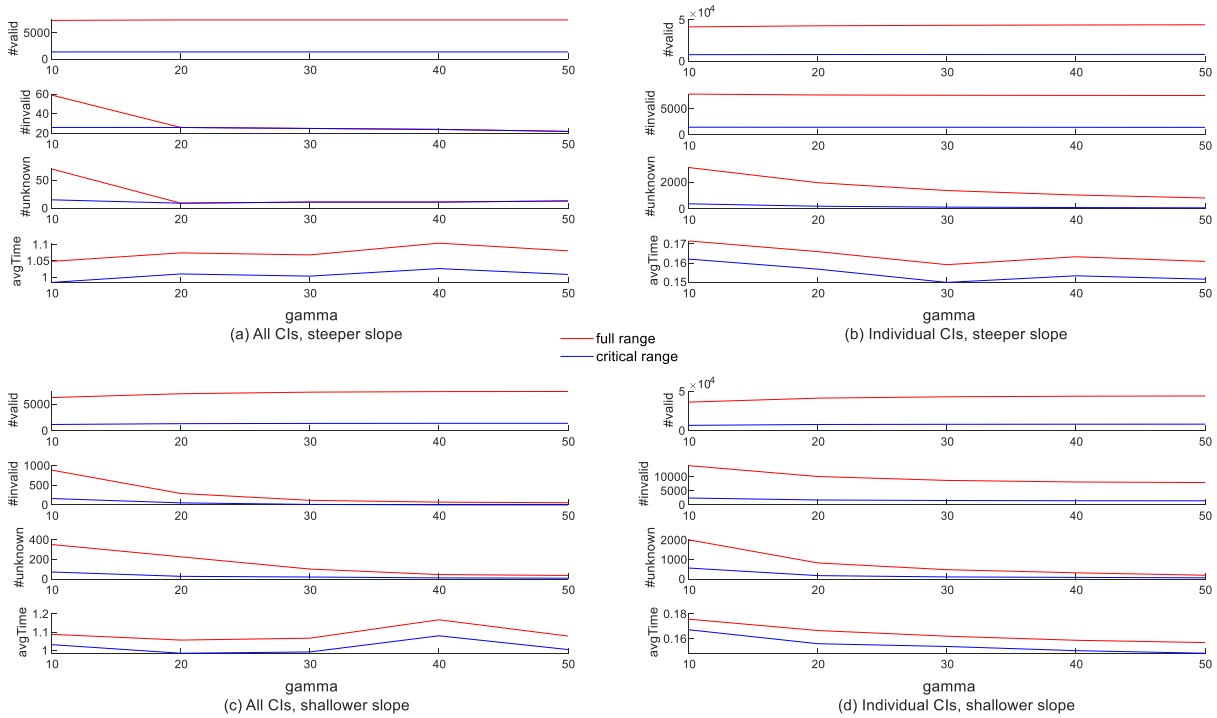

Figure 32. *Statistics on the property validity and average solving time for Phase 4 monotonicity analysis.*

Same as for stability properties, monotonicity verification can generate counterexamples for invalid properties. Such CEX can be fed back to SME for further analysis. An example is shown in ***Figure 33*** for one of the input windows in the critical region of the RUL. Here, the growth rate of all seven CIs has been increased by γ = 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 steps. The counterexample has been provided to SME for further investigation.

Overall, verification of local monotonicity properties defined from requirements RUL-ML-Mon-1 and RUL-ML-Mon-2 reveals that current version of the ML model does not meet either of the two requirements. This is due to:

- Significant number of property violations on relatively small changes of the CI growth rate (e.g., $\gamma = 10\%$) – 20% for all single-CI properties, with the majority of invalid properties belonging to the changes in ShaftOrder1 and ShaftOrder3 CIs. 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.

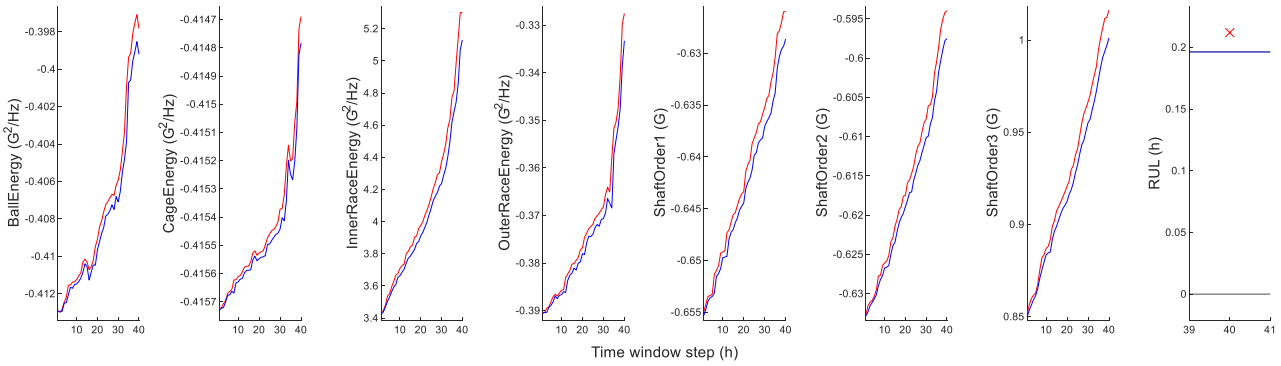

Figure 33. Counterexample for a monotonicity property with a 10% growth rate increase for all CIs.

Limited non-monotonicity. Results obtained from the monotonicity verification show that the behavior of the RUL estimator behavior is not fully monotonic: it is possible that it provides a higher RUL value, instead of a lower one in the presence of an increased growth rate of one or more condition indicators. 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, based on relaxations of (5.7a) and (5.8a) for, respectively, single and multiple CI growth rate increase (based on results in ***Table 15*** and ***Table 16***):

$$\forall \boldsymbol{x}' : \forall t : u_i^t \leq x_i'^t \leq \max(u_i^t, u_i^{t+1}) \Rightarrow f(\boldsymbol{x}') \leq f(\boldsymbol{x}) + \varepsilon \quad (5.9a)$$

$$\forall \boldsymbol{x}' : \forall i \in S : \forall t : u_i^t \leq x_i'^t \leq \max(u_i^t, u_i^{t+1}) \Rightarrow f(\boldsymbol{x}') \leq f(\boldsymbol{x}) + \varepsilon \quad (5.9b)$$

for step t, constant γ, constant $\varepsilon = 10$h[54], $u_i^t = x_i^t + \gamma|x_i^t - x_i^1|$, S being a subset of indices i of input features corresponding to CIs.

Equations (5.9a-b) impose that a constrained fluctuation in the CIs yields an increase of the estimated RUL that is limited by a constant number of hours ε. These analyses are complementary to (5.7a) and (5.8a), and

[54] This value of ε does not make part of current monotonicity requirements for the ML constituent. It has been defined based on SME feedback to the results of Phase 4 of the analysis.

are executed to assess limited non-monotonicity for all properties reported as invalid or unknown during Phase 4 verification (see above).

Results are summarized in ***Table 19*** and ***Table 20*** for, respectively, single CI growth rate increase (i.e., steeper slope) and all CIs simultaneous growth rate increase. In the former case, the majority of previously reported invalid/unknown properties become valid, i.e., the violation of the original property is bounded by $\varepsilon = 10h$. Noteworthy, there are few exceptions related exclusively to one of the CIs (BallEnergy). A small number of properties remains invalid, the analysis shows that they belong to a group of adjacent time windows in a critical range of one of the degradation sequences. The violations may be related to fluctuating behavior of the CI within the corresponding time windows. Possible reason may be related to errors in the simulations that produced synthetic data; it is not currently fully clear and needs to be further analyzed with the SME. In the latter case (growth rate change in all CIs) all previously violated properties become valid, i.e., also in this case the monotonicity property violation is bounded by $\varepsilon = 10h$.

Possible ML model improvements. One possible way of improving the monotonic behavior of the trained model is the modification of the learning process: training procedure could be guided towards solutions that preserve monotonicity more than others. The authors of [97] introduce a point-wise loss function for enforcing monotonicity in deep neural networks. The function incorporates traditional empirical risk minimization term, as well as additional terms that penalize non-monotonic behavior in the NN. Another approach, presented in [25], also modifies the learning process by making it iterative *counterexample-guided*. Training phases alternate with verification phases, where counterexamples for monotonicity properties are identified. These CEX are added to subsequent training phases, therefore, the overall approach is a type of an adversarial training method.

***Table 19**. Verification results for relaxed property that allows limited non-monotonicity (single CI, increased CI growth rates).*

setting	range	gamma	#prop	#valid	#invalid	#unknown	avgTime(s)	totTime(s)
2-step method **single CI, steeper slope, LIMITED NON-MON, epsilon=10h**	full	10%	10879	10879	0	0	0.1369	1489.97
		20%	9576	9576	0	0	0.1357	1299.40
		30%	8938	8938	0	0	0.1395	1247.25
		40%	8582	8572	9	1	0.1371	1176.95
		50%	8343	8332	11	0	0.1432	1194.60
	critical	10%	1837	1837	0	0	0.1342	246.45
		20%	1650	1650	0	0	0.1412	233.05
		30%	1582	1582	0	0	0.1468	232.32
		40%	1539	1529	9	1	0.1451	223.38
		50%	1511	1500	11	0	0.1489	224.9.4

***Table 20**. Verification results for relaxed property that allows limited non-monotonicity (all CIs, increased CI growth rates).*

setting	range	gamma	#prop	#valid	#invalid	#unknown	avgTime(s)	totTime(s)
2-step method all 7 CIs, steeper slope, **LIMITED NON-MON, epsilon=10h**	full	10%	129	129	0	0	1.113	143.60
		20%	35	35	0	0	0.8906	31.17
		30%	36	36	0	0	0.8382	30.17
		40%	35	35	0	0	0.8440	29.54
		50%	35	35	0	0	0.8652	30.28
	critical	10%	41	41	0	0	0.8422	34.53
		20%	35	35	0	0	0.8906	31.17
		30%	36	36	0	0	0.8382	30.17
		40%	35	35	0	0	0.8440	29.54
		50%	35	35	0	0	0.8652	30.28

To overcome ML model limitations in achieving a completely monotonic behavior, preservation of monotonicity could instead be ensured at the ML constituent level. A relevant approach of constructing a "monotonic envelope" and its further use as part of runtime monitoring to identify and correct non-monotonic behaviors has been discussed in Section 4.3.5.

5.4.5.2 Operational envelope impact

Desert operating environment is expected to place more stress on the bearing component than a non-desert (normal) one during the mission due to the increased presence of sand and dust in the atmosphere, which can penetrate the bearing and lead to faster degradation. The impact of the environment on the RUL should be observable on a sufficiently large time window. This aspect can be formulated as following properties:

$$\forall \boldsymbol{x}' : (\boldsymbol{x}_{env} = normal \wedge \boldsymbol{x}'_{env} = desert) \Rightarrow f(\boldsymbol{x}') \leq f(\boldsymbol{x}) \quad (5.9a)$$

$$\forall \boldsymbol{x}' : (\boldsymbol{x}_{env} = desert \wedge \boldsymbol{x}'_{env} = normal) \Rightarrow f(\boldsymbol{x}') \geq f(\boldsymbol{x}) \quad (5.9b)$$

where $\boldsymbol{x}'$ is the new time window derived from the original time window $\boldsymbol{x}$ by toggling the environment feature in the entire window[55] (from *normal* to *desert* or vice versa), $\boldsymbol{x}_{env}$ and $\boldsymbol{x}'_{env}$ are column vectors representing the environment feature in, respectively, $\boldsymbol{x}$ and $\boldsymbol{x}'$. Informally, for a given time window, executing the mission in a desert environment should yield a lower RUL than in a non-desert one.

Simulation-based method was used for verification of operational impact properties, since they only required a toggle of the Environment feature for all time steps of the input time window, all other features being fixed, to compare the estimator output for both operational environments. Results are shown in ***Table 21***.

Table 21*. Verification results for operational envelope impact properties.*

setting	range	#prop	#valid	#invalid	#unknown	avgTime(s)	totTime(s)
Sim-based method	full	7493	7414	79	0	0.029	217.92
Environment feature	critical	1414	1335	79	0	0.029	41.16

- Time window length of 40h is sufficient to observe the effect of the desert environment in terms of a lower estimated RUL compared to the normal environment.
- All invalid properties are in the critical range. This might suggest that, as the RUL of the bearing component approaches zero, operating environment impact on the RUL diminishes.

Possible ML model improvements. Additional examples of input windows corresponding to the critical range of the RUL, with same or similar values of all inputs except the Environment feature, and the RUL being smaller for the desert environment, may be added to the training dataset to improve the capability of the ML model to identify the impact of the operating environment on the estimation.

5.5 Results of applying formal methods on the RUL use case

This section discusses the results of the practical assessment of the use of formal methods on the remaining useful life use case, and the assessment results are contextualized with verification objectives for the RUL estimator. Demonstrated applications leverage formal methods and statistical methods and address several EASA AI Concept Paper objectives, including DM-13 (data quality verification), LM-10 (requirements-based

[55] Recall that, within a degradation sequence, and thus within any time window in that sequence, the environment does not change. Also the assumption stated in Section 5.4.3.1 holds: all other input features of x and x′ not explicitly mentioned in Equations (5.9a) – (5.9b) are assumed equal.

testing), LM-12 (trained model stability verification) and LM-13 (trained model robustness assessment)[56]. Key results are summarized below:

- Some features in the datasets have known distributions specified in the ML constituent ODD. These include condition indicators (Weibull distributions) and flight regime durations (truncated Normal distributions). Statistical methods provide efficient instruments to assess data representativeness with respect to such features. As a demonstrator, representativeness assessment of flight regime durations has been performed using goodness-of-fit testing (Sec. 5.3). Analysis results revealed that the durations of certain regimes in the data do not follow expected distributions[57]. Additionally, issues with data completeness for some flight regimes and missions have been identified. Results have been communicated to the SME to improve the quality of the datasets, e.g., to check the simulation models used to create synthetic data, and to generate more degradation sequences to improve data completeness.
- Several non-functional requirements specified in Section 2.2.4.2 have been expressed as formal properties that are amenable to formal verification by existing VNN tools. These include RUL estimator stability, monotonicity, and operational envelope impact.
- Requirements are formalized as global properties (for example, a global stability property is: "*For an input perturbation up to 40% near any input point in the ML constituent ODD, the output must not deviate by more than 10h*"). To overcome the limitations of FM in dealing with the verification of global properties of DNN and to provide evidence for the requirements, local properties have been formulated for each input point (time window) present in the test dataset. A necessary condition for such replacement of global properties with local ones is the high quality of the dataset in terms of completeness and representativeness with respect to ML constituent ODD.
- Local stability properties have been verified on the entire test dataset in two variants traceable to requirements RUL-ML-Stab-2 and RUL-ML-Stab-3: perturbations of a single CI and perturbations of all CIs. For the former, no invalid properties have been identified, i.e., requirement RUL-ML-Stab-2 is fully satisfied on the test data. For the latter, a small number of problematic inputs have been detected in some degradation sequences, where the property is violated or could not be proven by the FM tool. Overall, the number of violations over the test dataset is only 0.08%, while the number of unknown answers amounts to 1.58% of all inputs.
- Problematic inputs only occur in the beginning of some degradation sequences, i.e., on smaller CI values, when bearing health state is very high. No violations or unproven properties have been detected in the critical zone (with small values of RUL), where a possible safety impact could be present. This is an important observation since estimator stability in the critical zone has a relatively higher importance due to safety considerations (possible skip of a critical inspection of the bearing).
- For violated properties, counterexamples have been collected and analyzed. This revealed certain trends that could lead to ML model instability (e.g., simultaneous perturbation of several CIs by 40%). Such counterexamples have been provided to the SME for further investigation.
- Robustness assessment of the RUL estimator helped to understand the impact of larger perturbations than those prescribed by the model stability requirements, i.e., how much the ML model is robust to such unexpected fluctuations, how much its performance degrades. This analysis provided another important result: the number of invalid properties naturally grows with increased perturbation magnitude, but none of the violations ever occurs in the critical zone of the RUL (for CI

[56] Recall that ForMuLA project intends to demonstrate the applicability of FM for machine learning applications on a use case, not to demonstrate the full verification process to completely meet the assurance and certification objectives. Still, provided demonstrators focus on V&V activities that are critical and representative for the selected use case.
[57] Note that representativeness analysis must be conducted for each dataset separately (training, validation, test).

perturbations of up to 70%). This shows that the estimator is robust, particularly in the critical zone, where possible safety impact may be present.

- Verification of monotonicity properties has been carried out on the entire test dataset, with two variants of properties: growth rate (slope) increase of individual CIs and of all CIs, with an expected non-increasing value of the RUL. Verification revealed issues with monotonic behavior of the RUL estimator. In particular, invalid properties are present in the critical region of the RUL. Additionally, for small increases in the CI growth rate (e.g., 10%) the estimator often provided a higher RUL value, i.e., it was not able to correlate the small CI increment to increasing degradation. Counterexamples have been collected and communicated to the SME. A set of improvement suggestions for the ML model and constituent have been proposed, such as the use of a monotonic envelope as part of runtime monitoring, as well as the use of additional terms in the objective function of the training to penalize non-monotonic behavior.
- Operational envelope impact properties have been verified using simulation-based method. For most time windows from the test set, the estimator correctly incorporated the impact of the desert environment on the bearing degradation yielding a smaller RUL. Several violations have been identified, all of them related to time windows in the critical region of the RUL (last 100 hours). This reveals that in the critical RUL range other input features may be more relevant than the environment for the current ML model.
- Scalability issues of formal methods have been identified when using the exact (sound and complete) verification method. The method was successfully used to analyze local stability properties with perturbations applied only to a single condition indicator and provided verification results (still, the total time amounted to half a day for the available test dataset), but failed to scale beyond that. For example, for verification of stability properties with perturbations on multiple CIs, as well as for monotonicity properties, exact method did not provide any answer within the 3 hours timeout. Therefore, two-step verification method was used to trade off completeness for scalability of the analysis. This resulted in the verification of all properties on the entire test set, but provided a small percentage of *unknown* answers, i.e., some properties could not be neither proven nor disproven.
- Total execution time of the two-step method and simulation-based method has been reasonable for the available test dataset, ranging from several minutes to several hours for different analyses, i.e., stability, robustness, monotonicity, operational envelope impact. Verification time depended both on the number of properties and on the complexity of a single property that depends on the input space to be considered for the property. For example, stability property with single CI perturbation of a small magnitude can be verified on the order of seconds, while more complex property with multiple CI perturbations and larger magnitudes requires at least several minutes. Since properties are independent from each other, significant boost to the overall verification time can be achieved via parallelization of the analyses, for example, by leveraging cloud computing platforms.

5.6 Scalability and applicability assessment of formal methods

This section discusses the scalability and applicability aspects of formal verification methods provided by the learning assurance toolchain (exact, approximate, simulation-based, two-step), which is orthogonal to the verification of RUL use case requirements. These aspects have been quantitatively evaluated on a benchmark consisting of 1000 stability properties randomly picked from the test dataset for the RUL estimator. Multiple verification runs have been executed for each verification method while varying the complexity of stability properties. Stability property complexity is varied by including in the property formulation an increasing number of perturbed input window elements (e.g., multiple CIs, multiple time steps for a single CI) and varying the perturbation magnitude δ (10%, 40%, 60%), i.e., by modifying the input space related to the property. The intent is to observe the scalability, i.e., how execution time changes when property complexity is increased, and the applicability, i.e., the ability of providing a definitive answer to the verification problem (valid or invalid).

5.6.1 Experimental evaluation

Analysis results are summarized below:

a. **Exact verification method.** Results are reported in ***Figure 34***. It is evident that the verification time of the exact method grows exponentially when the number of perturbed input window elements is increasing, and soon becomes intractable. Approximately 3-5 perturbed input window elements, depending on δ, appear to be the empirical boundary on the applicability of the exact method on the RUL use case, and scalability limitations of the method are significant. As a baseline, execution time of the simulation-based method has been plotted (horizontal orange line). The baseline has been computed by generating 1000 modified time windows with random perturbations (bounded by δ) applied to CIs from the original time window, performing CNN inference for each of them, and repeating this for each property. Simulation-based method has constant time across the experiments, because it is only dependent on the number of fixed time windows to be evaluated, while inference time of the CNN is (almost) constant.

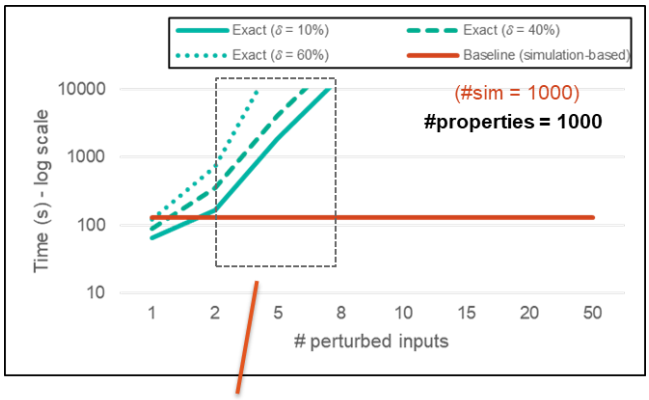

***Figure 34**. Scalability assessment of the exact verification method.*

b. **Approximate verification method.** ***Figure 35*** shows the percentage of benchmark properties that are proven valid using the approximate verification method, for a different number of perturbed input window elements and different perturbation magnitudes. For small δ the method is very efficient, as it can prove most of the properties, even when very many inputs fluctuate. The percentage drops significantly with the increasing δ, and with number of perturbations beyond 10. If many input elements are perturbed, the method may be able to prove less than a half of properties and returns an *unknown* answer for the rest. As further discussed in (d), approximate method exhibits better scalability *w.r.t.* exact method, however, it also has applicability limitations, because for high-complexity properties the number of unknown answers (unproven properties) may grow significantly.

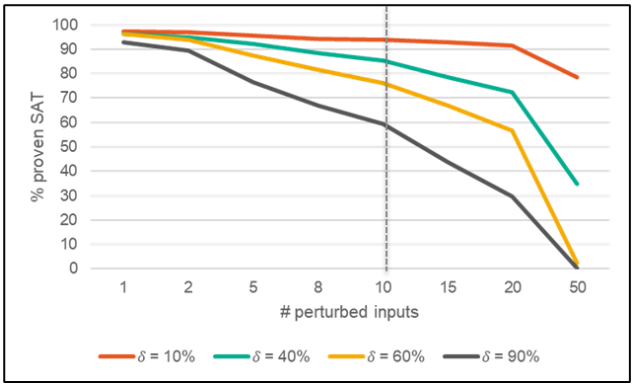

***Figure 35**. Percentage of stability properties proven valid by the approximate method (SAT = valid).*

c. **Simulation-based method.** ***Figure 36*** Illustrates the effectiveness of the simulation-based approach in formally *disproving*[58] properties, i.e., generating formal counterexamples, with respect to the number of perturbed inputs and magnitudes δ. The higher the complexity of the stability property, the more effective the method becomes in finding counterexamples. This can be explained by the fact that large perturbation magnitudes and/or number of perturbed inputs lead to a higher probability of violating the property, if the admissible deviation ε is fixed. If generation of random perturbations is done, e.g., by using a uniform distribution, then the probability of drawing an input from input space region that will lead to exceeding ε becomes significant as well. The plot also shows the effect of increasing the number of simulations by an order of magnitude (from 1000 to 10000), but mostly a minor increase in the percentage of disproven properties is observed as a result. Scalability of the simulation-based method does not depend on the complexity of the property, but rather on the number of random trials to be evaluated; as shown in ***Figure 34*** and ***Figure 37***, it requires constant time per simulation.

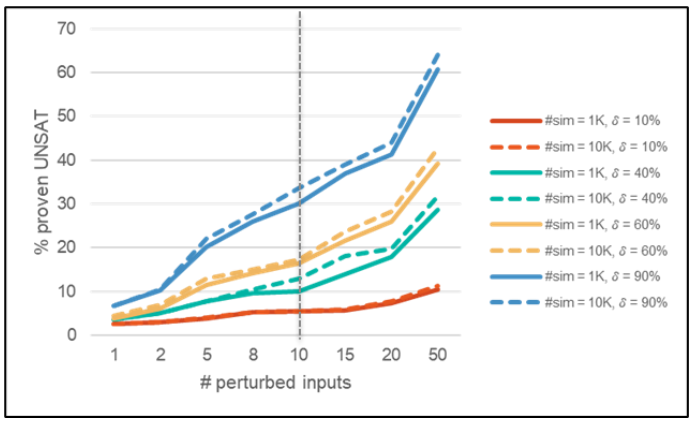

***Figure 36**. Percentage of stability properties proven invalid by the sim-based method (UNSAT = invalid).*

d. **Two-step verification method.** Same experiments as in (a) have been conducted for the two-step method. Results are shown in ***Figure 37***. The method still exhibits exponential growth[59], but can analyze more input perturbations, even at large magnitudes. For a small number of perturbed input window elements, the method even outperforms the baseline (simulation-based method) in terms of analysis time, because approximate reachability analysis (first step of the method) is executed

[58] As discussed in Section 5.2.2.2, this method is not able to formally prove the property (not being able to find a violation on a set of random trials is rather a statistical measure of validity, not a formal guarantee).
[59] Note that the time (Y axis) has a logarithmic scale, therefore, linear growth on this plot is in fact exponential.

faster than running 1000 simulations on that input (baseline), and each property that is proven valid with approximate reachability no longer requires simulation-based verification. With more perturbations, more unknown answers are produced by the approximate verification, and corresponding inputs need to be verified with simulation (second step of the method) in an attempt to falsify them. Therefore, the overhead increases, but it is still comparable with using only the approximate method for a number of perturbations less than 20. Larger amounts of perturbed inputs are rather synthetic (not realistic) for the current RUL use case.

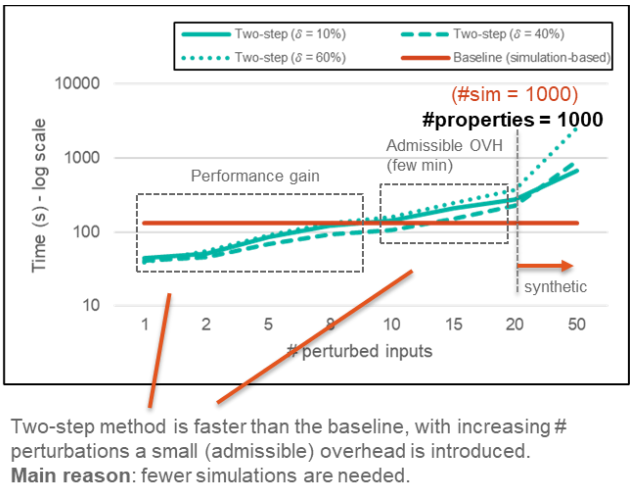

***Figure 37**. Scalability assessment of the two-step verification method.*

5.6.2 Assessment results

Numerical evaluation has been carried out on a benchmark consisting of a neural network architecture of average complexity (~100K learnable parameters), and a set of local stability properties of different complexity, which allowed to observe limitations of available formal verification approaches. Following key results can be highlighted:

- Applicability of the verification techniques has been assessed mainly on the complexity of the properties, i.e., on the size of the input space for which possible outputs (or output reachable set) had to be computed. The size of the input space is controlled by property parameters, such as perturbation magnitudes δ for local stability properties, as well as by the number of inputs on which the perturbations δ are applied. Verification techniques included exact and approximate formal analyses based on reachability and abstract interpretation, simulation-based, and hybrid approaches. Overall, the methods are applicable, but scalability limitations exist.
- Exact verification method performs well on properties with relatively small input space, i.e., involving perturbations applied to single or very few input window elements, and with smaller perturbation magnitudes (δ). However, as the input space grows (number of perturbed input window elements and/or δ is increasing), exact verification does not scale.
- Approximate verification method scales better on properties that consider larger input spaces, i.e., involving multiple perturbations and larger δ, but for a large number of perturbed input window elements its performance also deteriorates and eventually becomes impractical. Moreover, the higher the complexity of the property, the fewer properties are proven valid – the number of unknown answers grows significantly, which does not allow to make a definitive conclusion on the property validity.
- Simulation-based approach has performance that does not depend on the complexity of the property. By generating random bounded perturbations for the elements of the original time window related to the property it allows to identify counterexamples that lead to property violation, i.e., to

falsify the property. In case a CEX is found the property can be concluded invalid. Otherwise, no formal guarantee can be provided, and unknown answer is returned.

- Hybrid two-step technique involving approximate verification and simulation is an efficient way of improving the FM performance both in terms of execution time and the number of proven/disproven properties (reducing the number of unknown answers).
- For properties that involve large number of input perturbations (e.g., multiple input window elements over multiple time steps) and/or large perturbation magnitudes δ, both exact and approximate verification methods face scalability limitations. In this case, simulation-based approach could be used to provide a statistical argument about property validity (not a formal guarantee).
- Additional complexity of verification arises from the total number of properties to be verified. Even if a single property can be verified in reasonable time (e.g., several minutes), this will not be the case for thousands or millions of similar properties. This factor must be considered when the applicability of FM on a particular use case is assessed[60].

5.6.3 VNN competition and RUL benchmark

The CNN model of the RUL estimator discussed in this project has also been submitted as a benchmark to the 2022 Verification of Neural Networks Competition[61] (VNN-COMP'22), an event that aims to bring together researchers interested in methods and tools providing guarantees about the behaviors of neural networks and systems built from them. The competition is held annually to facilitate the fair and objective comparison of state-of-the-art NN verification tools and encourage the standardization of tool interfaces. In the 2022 iteration, 11 different tools participated on a diverse set of 12 scored benchmarks [68].

The RUL estimator CNN model from Collins Aerospace Applied Research & Technology has been provided as an ONNX model along with a set of properties specified in the VNNLib format (both formats are discussed in Section 3.5.3). Benchmark properties included some examples of local stability and local monotonicity properties, with different complexity. Despite being a deep learning model, the RUL estimator model had average complexity compared with other DNN models that participated in the competition as benchmarks, many of them focusing on image classification and relevant applications. Instead, Collins benchmark[62] suggested a prognostics application which, along with being one of the first industry benchmarks of VNN-COMP, would help to attract attention from VNN tool providers to address the needs of aerospace applications. In particular, RUL benchmark focused more on the *property complexity* as an important challenge for VNN tools scalability.

Results of the competition allowed to identify several VNN tools that demonstrated best performance on the RUL benchmark problem. Additional details can be found on the VNN-COMP website [59] and report [68].

[60] Use of high-performance computing (e.g., cloud infrastructures) and parallelization of verification on multiple workers could be a mitigation for this problem.
[61] https://sites.google.com/view/vnn2022
[62] Benchmark description, models and properties are available at https://github.com/loonwerks/vnncomp2022

6 Main conclusions of the project

Trustworthiness of machine learning enabled systems is an open problem, and barriers to their certification still exist. To overcome them, new assurance methods and processes need to be developed and adopted in the aviation industry. To make a step towards this goal, the ForMuLA project covered (1) an overview of state-of-the-art of ML-specific formal methods technologies and their limitations, (2) efficient ways of applying promising FM for ML development and V&V activities with a specific focus on the stability and robustness of machine learning models, and (3) identification of a subset of formal methods approaches as efficient means of compliance for certification objectives from the EASA AI Concept Paper.

The project consisted of two parts:

1. **Theoretical part** provided a comprehensive overview of formal methods technologies that have been adapted to or specifically developed for machine learning. Based on this overview, it then discussed possible applications of formal methods to ML-specific development and V&V activities. While key FM applications have been identified for the verification of ML models (e.g., property verification), various other innovative applications have also been discussed, including improvements of the learning process, generalization capability, and explainability. Additionally, statistical methods and their applications to data quality assessment and property verification have been discussed. For each class of applications, relevant objectives from the EASA AI Concept Paper have been identified, where formal methods could potentially contribute.
2. **Experimental part** was dedicated to a practical demonstration of applying formal methods and supporting statistical methods on a realistic use case of a deep learning-based estimator for remaining useful life of mechanical bearings for on-ground maintenance applications. Demonstrators covered a number of V&V activities within several ML development processes, namely Data Management (completeness and representativeness of the data) and Learning Process Verification, including the verification of stability and certain aspects of robustness of the ML model, such as identification of adversarial cases, as well as several other properties defined from non-functional requirements. Experimental results prove that formal methods could be effective means to support a number of critical verification objectives.

Based on both theoretical and experimental studies, the project has come to the following main conclusions:

1. Conclusions on available formal methods technologies for machine learning

a. Formal methods for machine learning are growing and maturing at a high pace, with the focus of the research community being on verification of neural networks. Several conventional techniques have been adapted to tackle ML verification problems, and novel tools are being developed to support them.
b. Current FM tools for ML are able to verify properties that impose relationships between ML model inputs and outputs. Several key ML properties, such as stability, can be expressed in this form and, therefore, supported by the FM tools.
c. Formal methods face scalability limitations that depend on the method, as well as on the model complexity and property complexity. While the use of sound and complete methods may be impractical for certain applications, other FM use approximations to trade off completeness and gain performance. Hybrid verification techniques (e.g., a combination of FM with simulation-based techniques) could also be effective means to improve the scalability of the verification problem.
d. FM technologies for ML can be applied beyond verification, but their effectiveness and limitations have to be better understood (see Future Work below).

2. Conclusions on the efficiency of formal methods for machine learning applications

a. FM can exhaustively verify regions of inputs of ML models and provide formal guarantees, thus ensuring higher confidence with respect to simulation-based verification techniques.
b. FM currently have scalability limitations in addressing global properties, in particular, for high-complexity ML models, such as deep learning. To mitigate this limitation, a reasonable approximation would be to instead verify a set of local properties on a test dataset. Completeness and representiveness of the dataset with respect to the ML constituent ODD is a necessary condition to replace global property analysis with a set of local properties.
c. Even for local properties, formal methods may not be able to provide complete verification due to the complexity of the ML model, or of the properties, or both. In such cases, hybrid verification procedures (e.g., combining FM with simulation-based methods) currently seem to be the most efficient way for improving verification completeness.
d. Overall number of properties can be extremely large. Total verification time must be considered when choosing to opt for formal verification. Even if the time to prove a single property is reasonable, the total time may not be, preventing the deployment at scale of the FM-based verification procedure.
e. Statistical methods are effective means to support objectives linked to data quality for known distributions.

3. Conclusions on effective means of compliance for ML

a. FM are effective means to support verification objectives linked to model stability, robustness (e.g., adversarial case identification, robustness tests), and intended behavior verification, as demonstrated on the RUL use case.
b. Currently, formal methods do not directly support the verification of global properties of ML models. However, FM can enable thorough verification of such properties by verifying a set of local properties on the test dataset, provided that it meets data quality requirements.
c. Investigations in ForMuLA supported the update of definitions and clarification of objective LM-11 on learning algorithm and trained model stability. The objective has been split into two new objectives during transition to the new version of the EASA AI Concept Paper: objective LM-11 concerns the stability of the learning algorithm while objective LM-12 deals with trained model stability.

Future work

Results of the ForMuLA project suggest several potential directions to further explore the applicability of formal methods to the assurance of machine learning. While ForMuLA overviewed a variety of possible FM applications, practical demonstrators have focused primarily on the property verification of the trained ML model. Future work, therefore, could focus on:

- Further investigation of the means of achieving complete verification with formal methods, while mitigating scalability challenges pertaining to FM.
- Use of formal methods for high-complexity ML applications, such as vision-based systems.
- Evaluation of FM applicability to support learning assurance beyond trained model verification – for example, data preparation, data quality verification, learning process improvements, ODD coverage, and consistency between test scenarios and ODD.
- Formal methods applications in the ML model implementation phase, in particular, for the demonstration of property preservation between trained and inference models.
- Possible contributions of FM to AI explainability.

These investigations may result in new demonstrators of the use of formal methods in the development and V&V of ML constituents, and their possible recommendation as effective means of compliance to an extended number of certification objectives

References

[1] RTCA/DO-333, “Formal Methods Supplement to DO-178C and DO-278A,” 2011.

[2] EASA, “Concept Paper: Guidance for Level 1&2 Machine Learning Applications. Concept paper for consultation,” Feb 2023.

[3] Daedalean, EASA AI task force, “Concepts of Design Assurance for Neural Networks (CoDANN),” EASA, Cologne, 2020.

[4] EASA, “Artificial Intelligence Roadmap: A Human-Centric Approach to AI in Aviation,” EASA, 2020.

[5] RTCA/DO-178C, “Software Considerations in Airborne Systems and Equipment Certification,” 2011.

[6] SAE G-34 Artificial Intelligence in Aviation, “Artificial Intelligence in Aeronautical Systems: Statement of Concerns,” SAE International, 2021.

[7] Daedalean, EASA AI task force, “Concepts of Design Assurance for Neural Networks (CoDANN) II,” EASA, Cologne, 2021.

[8] DEEL Certification Working group, “Machine Learning in Certified Systems,” 2020.

[9] F. Kaakai, K. Dmitriev, S. Adibhatla, E. Baskaya, E. Bezzecchi, R. Bharadwaj, B. Brown, G. Gentile, C. Gingins, S. Grihon and e. al, “Toward a machine learning development lifecycle for product certification and approval in aviation,” *SAE International Journal of Aerospace,* vol. 15, 2022.

[10] EASA, “Concept Paper: First usable guidance for Level 1 machine learning applications,” EASA, 2021.

[11] M. Pecht and J. Gu, “Physics-of-failure-based prognostics for electronic products,” *IEEE Transactions of Measurement and Control,* vol. 31, no. 3-4, pp. 309-322, 2009.

[12] “Similarity-Based Remaining Useful Life Estimation,” MathWorks, 2021. [Online]. Available: https://www.mathworks.com/help/predmaint/ug/similarity-based-remaining-useful-life-estimation.html.

[13] T. Benkedjouh, K. Medjaher, N. Zerhouni and S. Rechak, “Remaining useful life estimation based on nonlinear feature reduction and support vector regression,” *Engineering Applications of Artificial Intelligence,* vol. 26, no. 7, pp. 1751-1760, 2013.

[14] M. Yuan, W. Yuting and L. Li, “Fault diagnosis and remaining useful life estimation of aero engine using LSTM neural network,” in *IEEE international conference on aircraft utility systems (AUS)*, 2016.

[15] L. Ren, J. Cui, Y. Sun and X. Cheng, “Multi-bearing remaining useful life collaborative prediction: A deep learning approach,” *Journal of Manufacturing Systems,* vol. 43, pp. 248-256, 2017.

[16] X. Li, Q. Ding and J.-Q. Sun, “Remaining useful life estimation in prognostics using deep convolution neural networks,” *Reliability Engineering & System Safety,* pp. 1-11, 2018.

[17] “Remaining Useful Life Estimation using Convolutional Neural Network,” MathWorks, 2021. [Online]. Available: https://www.mathworks.com/help/releases/R2021a/predmaint/ug/remaining-useful-life-estimation-using-convolutional-neural-network.html.

[18] T. Mitchell, Machine Learning, New York: McGraw-Hill, 2007.

[19] EASA, “Notice of Proposed Amendment 2022-03: Reduction in accidents caused by failures of critical rotor and rotor drive components through improved vibration health monitoring systems,” EASA, 2022.

[20] L. de Moura and N. Bjorner, Formal methods: Foundations and applications, Springer, 2009.

[21] S. A. Seshia, A. Desai, T. Dreossi, D. J. Fremont, S. Ghosh, E. Kim, S. Shivakumar, M. Vazquez-Chanlatte and X. Yue, “Formal specification for deep neural networks,” in *International Symposium on Automated Technology for Verification and Analysis*, 2018.

[22] T. Dreossi, S. Ghosh, A. Sangiovanni-Vincentelli and S. A. Seshia, “A formalization of robustness for deep neural networks,” *arXiv preprint arXiv:1903.10033,* 2019.

[23] K. Gupta, B. Pesquet-Popescu, F. Kaakai and J.-C. Pesquet, “A quantitative analysis of the robustness of neural networks for tabular data,” in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021.

[24] A. Virmaux and K. Scaman, “Lipschitz regularity of deep neural networks: analysis and efficient estimation,” *Advances in Neural Information Processing Systems,* vol. 31, 2018.

[25] A. Sivaraman, G. Farnadi, T. Millstein and G. Van den Broeck, “Counterexample-guided learning of monotonic neural networks,” *Advances in Neural Information Processing Systems,* vol. 33, pp. 11936-11948, 2020.

[26] S. A. Seshia, A. Desai, T. Dreossi, D. J. Fremont, S. Ghosh, E. Kim, S. Shivakumar, M. Vazquez-Chanlatte and X. Yue, “Formal specification for deep neural networks,” *International Symposium on Automated Technology for Verification and Analysis,* 2018.

[27] C. Urban and A. Mine, “A review of formal methods applied to machine learning,” *arXiv preprint arXiv:2104.02466,* 2021.

[28] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barrett and M. J. Kochenderfer, “Algorithms for verifying deep neural networks,” *Foundations and Trends in Optimization,* vol. 4, pp. 244-404, 2021.

[29] R. R. Zakrzewski, “Fuel volume measurement in aircraft using neural networks,” in *International Joint Conference on Neural Networks*, 2001.

[30] A. Claviere, E. Asselin, C. Garion and C. Pagetti, “Safety verification of neural network controlled systems,” in *51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2021.

[31] M. H. Meng, G. Bai, S. G. Teo, Z. Hou, Y. Xiao, Y. Lin and J. S. Dong, “Adversarial Robustness of Deep Neural Networks: A Survey from a Formal Verification Perspective,” *IEEE Transactions on Dependable and Secure Computing,* 2022.

[32] C. Barrett and C. Tinelli, “Satisfiability Modulo Theories,” in *Handbook of model checking*, Springer, 2018, pp. 305-343.

[33] L. M. de Moura and N. Bjørner, “Z3: An Efficient SMT Solver,” in *Proceedings of the Theory and practice of software, 14th international conference on Tools and algorithms for the construction and analysis of systems (TACAS'08/ETAPS'08)*, Budapest, Hungary, 2008.

[34] H. Barbosa, C. Barrett, M. Brain, G. Kremer, H. Lachnitt, M. Mann, A. Mohamed, M. Mohamed, A. Niemetz, A. Nötzli and others, “cvc5: A Versatile and Industrial-Strength SMT Solver,” in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS) 2022*, Munich, Germany, 2022.

[35] G. Katz, C. Barrett, D. L. Dill, K. Julian and M. J. Kochenderfer, “Reluplex: An efficient SMT solver for verifying deep neural networks,” in *International conference on computer aided verification (CAV)*, 2017.

[36] L. Pulina and A. Tacchella, “An Abstraction-Refinement Approach to Verification of Artificial Neural Networks,” in *Computer Aided Verification. CAV 2010*, Edinburgh, UK, 2010.

[37] R. Ehlers, “Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks,” in *15th International Symposium on Automated Technology for Verification (ATVA)*, Pune, India, 2017.

[38] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljic and e. al, “The Marabou framework for verification and analysis of deep neural networks,” in *International Conference on Computer Aided Verification (CAV)*, 2019.

[39] N. Sato, K. Hironobu, Y. Nakagawa and H. Ogawa, “Formal Verification of a Decision-Tree Ensemble Model and Detection of Its Violation Ranges,” *IEICE TRANSACTIONS on Information and Systems,* pp. 363-378, 2020.

[40] G. Einziger, M. Goldstein, Y. Sa'ar and I. Segall, "Verifying Robustness of Gradient Boosted Models," in *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, Honolulu, Hawaii, USA, 2019.

[41] D. Gopinath, H. Converse, C. Pasareanu and A. Taly, "Property Inference for Deep Neural Networks," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019.

[42] S. Kolb, S. Teso, A. Passerini and L. De Raedt, "Learning SMT (LRA) constraints using SMT solvers," in *IJCAI International Joint Conference on Artificial Intelligence*, 2018.

[43] R. Evans and E. Grefenstette, "Learning explanatory rules from noisy data," *Journal of Artificial Intelligence Research,* vol. 61, pp. 1-64, 2018.

[44] R. Sebastiani and S. Tomasi, "Optimization modulo theories with linear rational costs," *ACM Transactions on Computational Logic (TOCL),* vol. 16, no. 2, pp. 1--43, 2015.

[45] V. Tjeng, K. Xiao and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," *arXiv preprint arXiv:1711.07356,* 2017.

[46] I. Grossmann, "Review of Nonlinear Mixed-Integer and Disjunctive Programming Techniques," *Optimization and Engineering,* vol. 3, p. 227–252, 2002.

[47] R. R. Bunel, I. Turkaslan, P. Torr, P. Kohli and P. K. Mudigonda, "A unified view of piecewise linear neural network verification," *Advances in Neural Information Processing Systems,* vol. 31, 2018.

[48] S. Dutta, S. Jha, S. Sankaranarayanan and A. Tiwari, "Output range analysis for deep feedforward neural networks," in *NASA Formal Methods Symposium*, 2018.

[49] X. Liu, X. Han, N. Zhang and Q. Liu, "Certified monotonic neural networks," *Advances in Neural Information Processing Systems,* vol. 33, pp. 15427-15438, 2020.

[50] M. Jordan and A. G. Dimakis, "Exactly computing the local lipschitz constant of relu networks," *Advances in Neural Information Processing Systems,* vol. 33, pp. 7344-7353, 2020.

[51] T. P. Pawlak and K. Krawiec, "Automatic synthesis of constraints from examples using mixed integer linear programming," *European Journal of Operational Research,* vol. 261, no. 3, pp. 1141-1157, 2017.

[52] E. A. Schede, S. Kolb and S. Teso, "Learning linear programs from data," in *IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, 2019.

[53] H.-D. Tran, X. Yang, D. Manzanas Lopez, P. Musau, L. V. Nguyen, W. Xiang, S. Bak and T. T. Johnson, "NNV: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems," in *International Conference on Computer Aided Verification (CAV)*, 2020.

[54] S. Wang, K. Pei, J. Whitehouse, J. Yang and S. Jana, "Efficient formal safety analysis of neural networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[55] S. Wang, K. Pei, J. Whitehouse, J. Yang and S. Jana, "Formal security analysis of neural networks using symbolic intervals," in *27th USENIX Security Symposium*, 2018.

[56] P. Henriksen and A. Lomuscio, "Efficient neural network verification via adaptive refinement and adversarial search," in *ECAI*, 2020.

[57] K. D. Julian, J. Lopez, J. S. Brush, M. P. Owen and M. J. Kochenderfer, "Policy compression for aircraft collision avoidance systems," in *IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, 2016.

[58] R. Bunel, P. a. T. I. Mudigonda, P. Torr, J. Lu and P. Kohli, "Branch and bound for piecewise linear neural network verification," *Journal of Machine Learning Research,* vol. 21, 2020.

[59] "3rd International Verification of Neural Networks Competition (VNN-COMP'22)," 2022. [Online]. Available: https://sites.google.com/view/vnn2022 .

[60] H. Zhang, M. Shinn, A. Gupta, A. Gurfinkel, N. Le and N. Narodytska, “Verification of recurrent neural networks for cognitive tasks via reachability analysis,” in *ECAI*, 2020.

[61] S. Bak, “nnenum: Verification of relu neural networks with optimized abstraction refinement,” in *NASA Formal Methods Symposium*, 2021.

[62] G. Singh, T. Gehr, M. Püschel and M. Vechev, “An abstract domain for certifying neural networks,” *Proc. ACM Program. Lang.,* vol. 3, p. Article 41, 2019.

[63] S. Wang, Z. Huan, X. Kaidi, L. Xue, J. Suman Sekhar, H. Cho-Jui and K. J. Zico, “Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Neural Network Robustness Verification,” in *Proceedings of the 35th International Conference on Neural Information Processing Systems*, 2021.

[64] H.-D. Tran, S. Bak, W. Xiang and T. T. Johnson, “Verification of deep convolutional neural networks using imagestars,” in *International conference on computer aided verification (CAV)*, 2020.

[65] F. Ranzato and M. Zanella, “Robustness verification of support vector machines,” in *International Static Analysis Symposium*, 2019.

[66] E. Wong and Z. Kolter, “Provable defenses against adversarial examples via the convex outer adversarial polytope,” in *International Conference on Machine Learning*, 2018.

[67] K. Xu, H. Zhang, S. Wang, Y. Wang, S. Jana, X. Lin and C.-J. Hsieh, “Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers,” *arXiv preprint arXiv:2011.13824,* 2020.

[68] M. N. Müller, C. Brix, S. Bak, C. Liu and T. T. Johnson, “The Third International Verification of Neural Networks Competition (VNN-COMP 2022): Summary and Results,” 2022.

[69] P. L. Combettes and J.-C. Pesquet, “Lipschitz Certificates for Layered Network Structures Driven by Averaged Activation Operators,” *SIAM Journal on Mathematics of Data Science,* vol. 2, pp. 529-557, 2020.

[70] T. Chen, J. B. Lasserre, V. Magron and E. Pauwels, “Polynomial Optimization for Bounding Lipschitz Constants of Deep Networks,” ArXiv preprint abs/2002.03657, 2020.

[71] M. Fazlyab, A. Robey, H. Hassani, M. Morari and G. J. Pappas, “Efficient and Accurate Estimation of Lipschitz Constants for Deep Neural Networks,” in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019.

[72] “Open Neural Network Exchange (ONNX),” 2022. [Online]. Available: https://onnx.ai/.

[73] “VNN-LIB standard for benchmarks,” 2021. [Online]. Available: http://www.vnnlib.org/index.html.

[74] D. Shriver, S. Elbaum and M. B. Dwyer, “DNNV: A framework for deep neural network verification,” in *International Conference on Computer Aided Verification (CAV)*, 2021.

[75] C.-H. Cheng, C.-H. Huang, T. Brunner and V. Hashemi, “Towards safety verification of direct perception neural networks,” in *Design, Automation and Test in Europe Conference (DATE)*, 2020.

[76] C.-H. Cheng and R. Yan, “Continuous Safety Verification of Neural Networks,” in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2021.

[77] M. A. Pimentel, D. A. Clifton, L. Clifton and L. Tarassenko, “A review of novelty detection,” *Signal processing,* vol. 99, pp. 215-249, 2014.

[78] C. Huber-Carol, N. Balakrishnan, M. Nikulin and M. Mesbah, Goodness-of-fit tests and model validity, Springer Science and Business Media, 2012.

[79] J. Antony, Design of experiments for engineers and scientists, Elsevier, 2014.

[80] V. L. Anderson and R. A. McLean, Design of experiments: a realistic approach, CRC Press, 2018.

[81] T. Ko, V. Peddinti, D. Povey and S. Khudanpur, “Audio augmentation for speech recognition,” *Sixteenth annual conference of the international speech communication association,* 2015.

[82] L. Taylor and G. Nitschke, "Improving deep learning with generic data augmentation," *2018 IEEE Symposium Series on Computational Intelligence (SSCI),* 2018.

[83] A. Antoniou, A. Storkey and H. Edwards, "Data augmentation generative adversarial networks," *arXiv preprint arXiv:1711.04340,* 2017.

[84] A. Gupta, L. M. Naman Shukla, A. Kolbeinsson and K. Yellepeddi, "How to incorporate monotonicity in deep networks while preserving flexibility?," *arXiv preprint arXiv:1909.10662,* 2019.

[85] X. Liu, X. Han, N. Zhang and Q. Liu, *Advances in Neural Information Processing Systems,* vol. 33, 2020.

[86] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang and L. Yang, "Physics-informed machine learning," *Nature Reviews Physics,* vol. 3, no. 6, pp. 422-440, 2021.

[87] M. Damour, F. D. Grancey, C. Gabreau, A. Gauffriau, J.-B. Ginestet, A. Hervieu, T. Huraux, C. Pagetti, L. Ponsolle and A. Claviere, "Towards Certification of a Reduced Footprint ACAS-Xu System: A Hybrid ML-Based Solution," in *International Conference on Computer Safety, Reliability, and Security*, 2021.

[88] R. Zakrzewski, "Verification of a trained neural network accuracy," in *International Joint Conference on Neural Networks*, 2001.

[89] K. Gupta, B. Pesquet-Popescu, F. Kaakai and J.-C. Pesquet, "A quantitative analysis of the robustness of neural networks for tabular data," *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),* 2021.

[90] W. Lin, Z. Yang, X. Chen, Q. Zhao, X. Li, Z. Liu and J. He, "Robustness verification of classification deep neural networks via linear programming," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition,* 2019.

[91] M. Balunovic and M. Vechev, "Adversarial training and provable defenses: Bridging the gap," in *International Conference on Learning Representations*, 2019.

[92] N. Dahlin, K. C. Kalagarla, N. Naik, R. Jain and P. Nuzzo, "Designing interpretable approximations to deep reinforcement learning," *arXiv preprint arXiv:2010.14785,* 2020.

[93] A. Sivaraman, G. Farnadi, T. Millstein and G. Van den Broeck, "Counterexample-guided learning of monotonic neural networks," *Advances in Neural Information Processing Systems 33,* 2020.

[94] J. Rushby, "Automated test generation and verified software," in *Working Conference on Verified Software: Theories, Tools, and Experiments*, 2005.

[95] C. Hutchison, M. Zizyte, P. E. Lanigan, D. Guttendorf, M. Wagner, C. L. Goues and P. Koopman, "Robustness testing of autonomy software," *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice,* 2018.

[96] ASTM F3269-17, "Standard practice for methods to safely bound flight behavior of unmanned aircraft systems containing complex functions," 2017.

[97] A. Gupta, N. Shukla, L. Marla, A. Kolbeinsson and K. Yellepeddi, "How to Incorporate Monotonicity in Deep Networks While Preserving Flexibility?," *arXiv preprint arXiv:1909.10662,* 2019.

[98] G. Katz, C. Barrett, D. L. Dill, K. Julian and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," *International conference on computer aided verification,* 2017.

[99] Y. Sun, X. Huang, D. Kroening, J. Sharp, M. Hill and R. Ashmore, "Structural test coverage criteria for deep neural networks," *ACM Transactions on Embedded Computing Systems (TECS),* vol. 18, no. 5s, pp. 1-23, 2019.

[100] SAE G-34; EUROCAE WG-114, "AIR6987: Artificial Intelligence in Aeronautical Systems: Taxonomy," SAE/EUROCAE.

[101] C.-H. Cheng, G. Nührenberg and H. Ruess, "Maximum resilience of artificial neural networks," in *International Symposium on Automated Technology for Verification and Analysis*, 2017.

Appendix 1: Entire spectrum of FM applications in the ML context

This Appendix discusses a broader scope of formal methods applications in the ML development lifecycle[63]. It first introduces an extended lifecycle diagram that is aligned with the W-cycle, as well as with available published efforts of SAE G-34/EUROCAE WG-114. It then discusses the traditional V&V objectives that can be addressed by FM, as per ED-216/DO-333, and gives an intuition of relevant ML applications in ML lifecycle processes. In particular, each process is reviewed in detail, and all possible V&V activities that may be addressed by formal methods are listed. Alignment with EASA AI Concept Paper objectives is also provided.

A1.1. Machine learning development lifecycle

Figure 38 shows, on a V-like diagram, development and verification activities for an ML constituent in a flow that moves from system requirements, system architecture, and operational design domain inputs to the implemented ML constituent – i.e., the final output. The figure illustrates processes as dotted rectangles, ML life cycle artifacts (data objects) as ovals, and activities as arrows. Each process consists of development and V&V parts, shown as solid rectangles. Each process takes lifecycle artifacts as inputs and possibly produces other artifacts as outputs. Solid line arrows capture the artifact flow among development activities, while dashed arrows capture the artifact flow for V&V activities. The diagram can be viewed as an extension of a traditional V-model for representing a system development life cycle. Here, the left-hand side of the V additionally includes processes and artifacts specific to ML, such as data management (collection, preparation) and learning process (part of the ML Model Design process), as well as V&V activities aimed at ML requirements, data, and models. The right-hand side of the V corresponds to integration, validation and verification of the ML constituent.

The figure is divided by two green dotted lines into three parts, respectively denoting system level, ML Constituent level (scope of this IPC), and item level. The reader is referred to standards ARP4754A, ED-12C/DO-178C and ED-80/DO-254, for guidance respectively at system/subsystem levels higher than ML constituent, and at SW and HW item levels, where traditional processes are expected to apply.

The lifecycle diagram illustrates several processes along the ML constituent development path. The ML ODD and Requirements process regards the definition of the ML constituent ODD and requirements allocated to the ML constituent. The ML Data Management process is the same as in the W-model and includes data collection and preparation (development activities), as well as data quality verification (V&V activities for data). ML Model Design process incorporates selection of the model architecture, training algorithm, hyperparameters, objective function and validation metrics, the process of training and the verification activities for the trained ML model. Finally, ML Constituent Implementation process deals with the creation of the inference model as well as other items that are part of the ML constituent (e.g., preprocessing and postprocessing) that could be based on traditional software development processes.

The proposed lifecycle offers a different perspective on the W-cycle proposed in [3] and further developed in the EASA AI Concept Paper [2]. It can be seen as an "unrolled" version of the W-shaped diagram that rather follows a "staircase" pattern on the left-hand side. Along with the ML model verification step specifically highlighted in the middle of the W-cycle, ***Figure 38****Figure 38* also emphasizes V&V steps for each of the development processes, i.e., going up performing a "small right-hand side" step. The key motivation of such unrollment and restructuring lies in the alignment with existing standards (ED-216/DO-333), as well as with the emerging ML development process that is currently work-in-progress in the SAE G-34 / EUROCAE WG-114 joint working group on AI/ML in aviation [9]. Elements of the W-model are also shown on the diagram.

[63] Some FM applications listed in this Appendix may not be ML-specific, that is, applicable to machine learning in the very same way as in traditional systems.

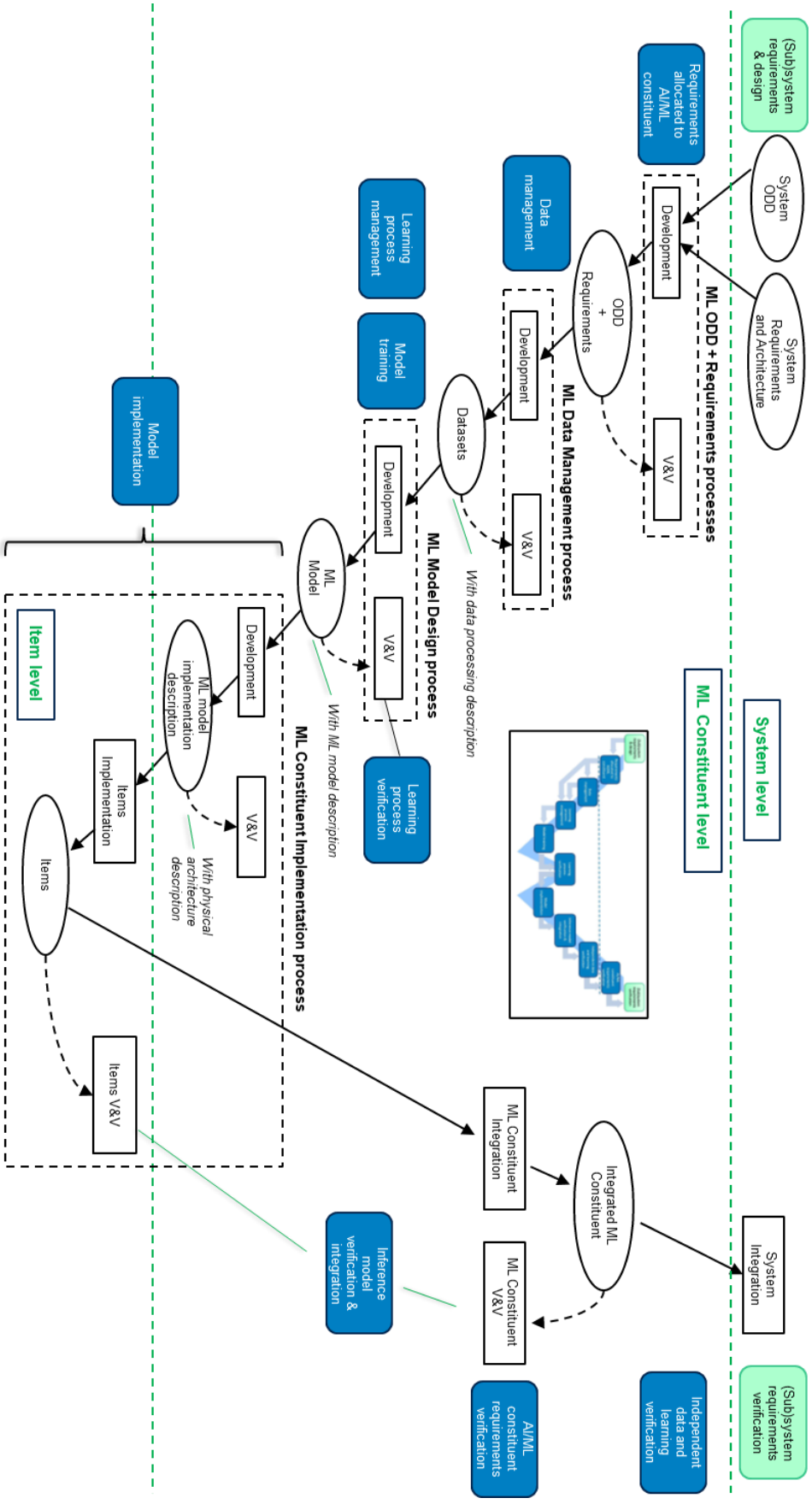

Figure 38. *ML Constituent development lifecycle aligned with the W-cycle.*

A1.2. V&V objectives for machine learning

V&V is typically performed by a combination of reviews, analyses and testing. The latter two are offering quantitative and repeatable evidence of correctness. Depending on whether the artifacts under verification admit a formal model, it is possible to make the V&V process partially or fully automated by relying on the use of formal analyses and formal methods.

The V&V activities are carried out to satisfy verification objectives of different types, associated with development processes and life cycle data objects. In the following, an overview of the types of verification objectives that *may* be addressed by means of formal methods is given. A major part of these objectives is adopted as in avionics standards (ED-12C/DO-178C; ED-216/DO-333), while some of them include ML-specific aspects and activities, which are highlighted.

A. Traceability

Traceability is defined as an association, typically bidirectional, between life cycle data objects.

Use of FM: In presence of formal models, traceability can be demonstrated by means of formal methods from the inputs to the outputs of a process, as well as from the outputs to the inputs [1]; compliance (see G below), for example, can provide evidence in support of traceability.

B. Compatibility

Compatibility refers to the lack of conflicts between two life cycle data objects.

Use of FM: If data objects are formally modeled, formal methods can be used to show compatibility between two such objects, including an object and a formal description of software and/or hardware features of the target computer.

C. Accuracy[64]

Accuracy of a life cycle data object refers to the adherence of the data object to its intended behavior or functionality.

Use of FM: The use of a formal notation, with a defined syntax and semantics, favours precision while preventing ambiguities, and can be used to show accuracy of a data object representation.

D. Consistency

Consistency of a collection/composition of life cycle data objects refers to the presence of harmony and agreement among the elements of the compound.

Use of FM: The adoption of a formal representation for the description of data objects allows to detect a lack of consistency in a given collection of such objects.

E. Verifiability

Verifiability of a life cycle data object refers to the possibility of determining whether a property of interest is valid or not for the data object.

Use of FM: If a data object is formally modeled, then the use of a mathematically defined syntax and semantics makes it amenable to formal verification.

[64] Note that accuracy in the sense of ED-12C/DO-178C and ED-216/DO-333 is a concept distinct from accuracy of an ML Model, which is instead an ML-specific performance metric like e.g. precision and recall.

F. Conformance to Standards

Conformance refers to the adherence of a life cycle data object to the standards defining its development process.

Use of FM: Data objects that are formally represented can be automatically checked for conformance with the formal notation by means of formal methods.

G. Compliance

Compliance refers to the realization of the intended relationship between inputs and outputs of a development process.

Use of FM: If the life cycle data objects of a process are formally modeled, formal methods can be used to provide verification evidence of compliance of one representation with the other. In particular, the output of a development process can be shown to satisfy properties derived from the process inputs; in case a property is violated, a counterexample is usually provided as evidence, and can be employed to refine the relevant life cycle data.

Compliance between Requirements

Example (Compliance between HLR and LLR). If a set of higher level requirements (HLR) and a set of lower level requirements (LLR), developed from the former, are at a comparable level of detail and formalization, then the compliance of the latter with the former can be shown by means of property verification, checking the implication:

$$llr_1 \wedge \ldots \wedge llr_n \Rightarrow hlr_1 \wedge \ldots \wedge hlr_m,$$

where $\{hlr_i\}$ and $\{llr_j\}$ are respectively the formal representations of higher and lower level requirements.

Compliance with a Property

Example (Functional correctness). A property expressed on the ML model inputs and outputs, for example of the form "if the inputs satisfy a condition P, then the outputs must satisfy a condition Q", can be subject to verification approaches, that return a counterexample showing a violation, in case the property is not valid ([26], [98]). The verifiability of such a property depends on the complexity of the model and of the complexity of the property, as discussed in Section 5.6.

H. Coverage

Requirements Coverage

Requirements coverage analysis is a verification activity executed to determine whether any requirement is missing verification evidence. A noteworthy case is represented by the expectation for the datasets to reach a sufficient coverage of the ML data requirements, based on which they are developed, including coverage of the ML constituent ODD.

Example (ODD coverage). Verification of coverage of ODD space by properties (both defined as n-dimensional polyhedra)[65], identification of uncovered ODD regions as counterexamples. If, for example, properties are expressed in the premise-conclusion form (e.g., "if the inputs satisfy a condition P, then the outputs must satisfy a condition Q"), it can be verified whether the union of the polyhedrals satisfying the premises covers the ODD ([87]).

[65] A coverage metric needs to be defined for the given ODD.

Example (ML Perturbation coverage). Verification of coverage of possible input perturbation space by stability and/or robustness properties. The goal is to answer the question: *Are the existing tests/properties sufficient to assess the stability/robustness of the ML model?*

A1.3. V&V applications of formal methods in the ML development lifecycle

This section provides detailed analysis of each individual process of the ML constituent development lifecycle discussed in Section A1.1. This information is commercial proprietary and is not available in the public version of the report.

Appendix 2. Requirements formalization for the RUL use case

In this Appendix, the formalization of requirements related to the ML constituent ODD and to the functional capabilities of the RUL estimator are discussed. It is then shown how to apply formal reasoning in view of the V&V objectives presented in Appendix 1.

At system level, VHS-1 introduces a function sys_RUL, with output sys_est_RUL, for estimating the remaining useful life of a bearing component while the aircraft is on the ground. According to VHS-2, VHS-3, and the data provided by the ML constituent ODD, the function has eight inputs corresponding to the condition indicators and the torque for the component, plus four additional inputs: flight regime, nominal load, mission, environment condition. Thus, a total of twelve inputs. VHS-4 identifies the inputs as sequences of values captured across time steps in a given time window.

Based on this information, the RUL estimation function can be expressed as:

$$sys\_est\_RUL(t) = sys\_RUL(< in_1^{t-L}, \dots, in_1^t >, \dots, < in_{12}^{t-L}, \dots, in_{12}^t >)$$

where in_j represents the j^{th} input feature for brevity, and $L + 1$ is the size of the time window with respect to the current time step t.

Requirements VHS-7 and VHS-8 define the tolerance to the discrepancy between the estimated RUL and the actual value from the source data, depending on whether the actual RUL is in the "normal" or "critical" range:

$$normal(act\_RUL) \Rightarrow sys\_est\_RUL \leq (1 + 30\%) * act\_RUL \wedge sys\_est\_RUL \geq (1 - 10\%) * act\_RUL$$

$$critical(act\_RUL) \Rightarrow sys\_est\_RUL \leq (1 + 5\%) * act\_RUL \wedge sys\_est\_RUL \geq (1 - 15\%) * act\_RUL$$

VHS-9 places a limit on the overall error of the estimator as the *Mean Absolute Error (MAE)*:

$$\sum_{t=1}^{N} \frac{|sys\_est\_RUL(t) - act\_RUL(t)|}{N} \leq 15h$$

where N is the number of samples in the dataset of reference.

At the ML Constituent level, an estimation function $cons\_est\_RUL = cons\_RUL(\dots)$ is introduced, that traces to sys_RUL at system level. The requirements are developed maintaining bi-directional *traceability* to system requirements, so that each ML Constituent requirement maps to one system requirement, and vice-versa each system requirement maps to one or more ML Constituent requirements.

RUL-ML-1, traced to VHS-1, specifies that the function has a single non-negative output $cons\_est\_RUL$ measured in hours.

RUL-ML-2 and RUL-ML-3 further develop VHS-2 and VHS-3, by distinguishing between numerical and categorical features, derived from the system inputs; each feature domain, as well as its measure unit and data type, is indicated by the ML constituent ODD requirements. A formal specification of function output and input data types (numeric/float, categorical/string, …) allows for an automated *conformance* check on the values in the source data file, as well as a *compatibility* check e.g. with respect to the numerical precision of the target machine where RUL estimation will be run.

RUL-ML-4 to RUL-ML-7 trace to VHS-4 in describing $cons\_RUL$ as having as single input a bidimensional matrix, where the matrix columns correspond to the individual inputs of sys_RUL, and the rows to the time steps. The size of the time window, i.e. the amount of time steps, is set to 40; the values of all input vectors are intended as associated to consecutive time steps of duration 1h in the source data, so that the last vectors values map to the last time step in the data. $cons\_RUL$ input can then be represented as:

$$\begin{pmatrix} in_1^{t-39h} & \cdots & in_{12}^{t-39h} \\ \vdots & \ddots & \vdots \\ in_1^{t} & \cdots & in_{12}^{t} \end{pmatrix}$$

Compliance between system and ML Constituent input requirements is shown e.g. by equating each sys_RUL input i to column i of the $cons\_RUL$ input matrix.

RUL-ML-8 and RUL-ML-9 directly trace to VHS-6 and VHS-7, which they refine.

RUL-ML-14 quantifies the "critical" range (and, consequently, the "normal" range), introduced in VHS-7 and VHS-8, as 100h; in turn, these requirements are respectively developed into RUL-ML-10, RUL-ML-11, and RUL-ML-12, RUL-ML-13, as follows:

$$act\_RUL > 100h \Rightarrow cons\_est\_RUL \leq (1 + 30\%) * act\_RUL$$

$$act\_RUL > 100h \Rightarrow cons\_est\_RUL \geq (1 - 10\%) * act\_RUL$$

$$act\_RUL \leq 100h \Rightarrow cons\_est\_RUL \leq (1 + 5\%) * act\_RUL$$

$$act\_RUL \leq 100h \Rightarrow cons\_est\_RUL \geq (1 - 15\%) * act\_RUL$$

Note that the use of a logic representation is a means to ensure *accuracy* in the description of the expected behavior of the RUL estimator; also, it makes the output *verifiable* against the formalized requirements.

The formalization above allows to determine *consistency* among the ML Constituent output requirements for the RUL estimator: it is possible, for example, to conjoin the four requirements encodings and feed the conjunction to an automated reasoning engine for a satisfiability check.

In a similar way, *compliance* between e.g. "normal" range output requirements at ML Constituent level and at system level can be shown by noting that:

$$act\_RUL > 100h \Leftrightarrow normal(act\_RUL)$$

and formally proving the implication:

$$\begin{gathered} \big((act\_RUL > 100h \Rightarrow cons\_est\_RUL \leq (1 + 30\%) * act\_RUL) \\ \wedge \quad (act\_RUL > 100h \Rightarrow cons\_est\_RUL \geq (1 - 10\%) * act\_RUL)\big) \\ \Rightarrow \\ (normal(act\_RUL) \Rightarrow sys\_est\_RUL \leq (1 + 30\%) * act\_RUL \wedge sys\_est\_RUL \geq (1 - 10\%) * act\_RUL) \end{gathered}$$

RUL-ML-15 refines VHS-9 by further lowering the admissible error, expressed via the RMSE:

$$\sqrt[2]{\sum_{t=1}^{N} \frac{\big(cons\_est\_RUL(t) - act\_RUL(t)\big)^2}{N}} \leq 15h$$

Compliance of RUL-ML-15 to VHS-9 is due to the fact that $MAE \leq RMSE$ by definition.