

KISS My ML Assurance

Darren Cofer
Collins Aerospace
Minneapolis MN
darren.cofer@collins.com

Dmitrii Kirov
Collins Aerospace
Trento Italy
dmitrii.kirov@collins.com

Arthur Clavière
Collins Aerospace
Toulouse France
arthur.claviere@collins.com

Heber Herencia-Zapana
Collins Aerospace
Albany NY
heber.herencia-zapana@collins.com

Kyle Ford
Collins Aerospace
Cedar Rapids IA
kyle.ford@collins.com

Scott Nagel
Collins Aerospace
Cedar Rapids IA
scott.nagel@collins.com

Cong Liu
Collins Aerospace
Folsom CA
cong.liu@collins.com

Abstract— For some aircraft systems, the use of machine learning (ML) technologies such as neural networks (NN) may be a desirable choice for implementing a software component. In this paper we advocate a “Keep it simple stupid” (KISS) approach to evaluation and approval of ML-based components as part of the aircraft certification process. We show that the use of ML technologies to implement all or part of a software component doesn’t change most of the traditional (meaning non-ML) development and assurance activities. However, there are some aspects of traditional activities that present challenges or do not provide the expected assurance, necessitating a different approach. We conclude by applying our approach to an example avionics system to illustrate the concepts.

Keywords—machine learning, assurance, certification.

I. INTRODUCTION

Keep it simple stupid — Clarence L. “Kelly” Johnson, Lockheed Chief Research Engineer

For some aircraft systems, the use of machine learning (ML) technologies such as neural networks (NN) may be a desirable choice for implementing a software component. In this paper we advocate a “Keep it simple stupid” (KISS) approach to evaluation and approval of ML-based components as part of airworthiness certification requirements. Specifically, we will show that the use of ML technologies to implement part of an item doesn’t change most of the traditional (meaning non-ML) item-level development and assurance activities and objectives. Our goal is to keep things as simple as possible by making maximal use of the existing terminology, objectives, and activities of DO-178C [1], showing how ML might be accommodated with the minimal amount of impact on traditional processes.

Why would any changes to traditional processes be necessary? The use of an example dataset to specify the desired function and the creation of that function implementation by means of a training algorithm present technical differences due to the assumptions made by traditional processes. In some cases, these processes do not provide the expected assurance and must be interpreted or accomplished differently for items containing ML-based components.

From a regulatory perspective, the FAA Transport Aircraft Issues List [2] requires special approval activity for the use of ML or artificial intelligence (AI) in software or airborne electronic hardware. Entry number T-1619 requires applicants

to coordinate with the FAA if they are planning to develop any AI or ML based technology in their design and states that a means of compliance issue paper may be needed to obtain approval for these emerging technologies.

In this paper, traditional software development refers to the use of DO-178C as a means of compliance with airworthiness regulations. DO-178C describes processes to detect (and hopefully prevent) development errors in software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function. Including ML-based components as part of that software may produce contributors to failure conditions that are not caused by traditional software development errors. We must show that ML-based components can be developed with the same level of confidence required for the rest of the traditional software components in the item.

Unlike the traditional software development methods around which DO-178C is based, neural networks consist of a collection of nodes (neurons) arranged in layers. In a basic feed-forward NN, neurons in each layer receive inputs from the previous layer, multiply those inputs by parameters called weights, sum the weighted inputs, apply a non-linear activation function (e.g., tanh or ReLU), and pass the result to the next layer. In supervised learning, the ML training process starts with a large set of example input values and expected outputs (the training dataset) representing the function to be approximated by the NN. After selecting the topology of the network and activation functions to be used, the training algorithm iteratively computes the values of the NN weights, eventually converging to a set of weights. The resulting trained network is called an *inference model*.

A distinguishing characteristic of machine learning is its reliance on data to define the behavior of the component to be developed. Instead of transforming abstract requirements into successively more concrete specifications through an engineering design process, ML training algorithms use the dataset to produce an inference model based on the specific examples provided by the data. The Operational Design Domain (ODD) for the inference model defines the input space where the model is intended to operate. The inference model generalizes based on its training data to compute what the desired behavior should be for inputs on which it has not been trained. The performance of the inference model is evaluated

over an independent test dataset relative to defined statistical acceptance criteria.

In DO-178C several of the activities performed are intended to provide confidence in the absence of *unintended function*. The purpose is to determine if there is any part of the implementation that has not been evaluated by the verification process (requirements-based test cases) that could still be triggered during operation and result in incorrect or unsafe behavior. In an ML-based component, unintended function could occur due to inadequate training or bad design choices for the NN architectural parameters. However, traditional assurance activities are inadequate for detecting and preventing unintended function in ML-based components.

For the purposes of this paper, we consider only NNs that are trained during the development process using supervised learning. This means that the inference model deployed in the aircraft as part of a software item is fixed and does not change during operation. It is therefore a deterministic function – the same input always yields the same output.

After considering some recent related work, we will describe the proposed KISS approach for software ML-components in more detail. We then map this approach to the objectives of DO-178C and describe where new interpretations or new activities would be needed. We conclude by applying the approach to an ML-based function for computing a recommended aircraft cruise level to minimize cost associated with time or fuel. We show how traditional objectives and activities should be interpreted and also demonstrate how a formal methods analysis can be used to provide confidence in the absence of unintended function.

II. RELATED WORK

This paper describes one way to apply DO-178C to the development of ML-based components in a software item, but it is certainly not the only way that this could be done. Furthermore, the approach described here is intended to make maximal reuse of the existing terminology, objectives, and activities of DO-178C, showing how ML might be accommodated with the minimal amount of impact on traditional processes.

Other recent works looking at ML development and assurance processes for aircraft have different focus areas or viewpoints. For example, the EASA Artificial Intelligence Concept Paper [3] conceives of ML-based applications as part of a larger machine learning constituent (MLC), defining processes and objectives in a new development layer residing between the system level and the item level. Its approach is specifically driven by the emerging and more expansive needs of autonomous systems, and includes details related to ethics, safety, and explainability of the MLC. Furthermore, it is initially limited in scope to only lower criticality systems.

Wasson and Voros [4] address both assurance and regulatory approval for the use of ML in aircraft, tracing concerns and gaps back to the system safety process and means of compliance. The paper advocates for using current engineering principles to design and assure ML items rather than creating entirely separate, non-compatible processes, which is very much aligned with the goals of this paper. However, their work is more

descriptive and does not attempt to align ML development with the specific objectives and activities of DO-178C.

Another closely aligned paper is Dimitriev, et al. [5] which investigates the assurance and approval requirements for low-criticality (Design Assurance Level or DAL D) systems. It identifies gaps between DO-178C and the nature of ML, especially those related to the difficulty of requirements-based traceability. They propose a new workflow that adapts existing standards to meet safety objectives for ML systems, focusing on data quality, model training, and verification, arguing that low-criticality ML systems can be certified under current regulations.

The most important assurance gap we deal with in this paper is the need to provide confidence in the absence of unintended function, and the fact that the methods provided to accomplish this goal in DO-178C (traceability and structural coverage analysis) don't provide the same assurance for ML-based components. The Overarching Properties described in [6] define actual assurance goals that should be achieved rather than specifying particular methods that must be used. For example, the Overarching Property labeled Innocuity states "Any part of the implementation that is not required by the defined intended behavior has no unacceptable impact." Our approach attempts to follow a similar path by defining a new objective related to unintended function, rather than specifying the methods that must be used.

III. APPROACH

This section describes our proposed alignment of the artifacts and activities involved in developing an ML-based component with the activities of DO-178C. For some item in an aircraft system architecture, we assume that ML has been selected as the method to implement part of its functionality. Specifically, the System Requirements allocated to the software item will be decomposed and a subset will become High Level Requirements (HLR) for a component in the item. These HLR will be implemented using ML.

In warning against combining HLR and Low Level Requirement (LLR) into a single level of requirements, DO-248C [7] states that HLR represent *what* is to be implemented and LLRs represent *how* to carry out the implementation. In this context, inputs to the ML training process will be HLR (what is to be implemented) and outputs of the ML training process will be LLR (how to carry out the implementation in the inference model).

The input space where the component is intended to operate is known as the operational design domain (ODD) and is defined based on the component HLR. A dataset is produced that describes by example the behavior specified by the component HLR. There are many ways this dataset can be produced, and the details are beyond the scope of this paper. This data is partitioned into a training dataset (which specifies the required behavior of the component) and a test dataset (which is used to verify the trained model). The training and test datasets are drawn from the ODD and must be disjoint. They should both provide adequate coverage of the ODD and have similar statistical distributions.

This approach creates multiple HLR at a lower level (more detailed) than the top-level component HLR. Since the ODD is

part of the definition of what the component should do, it is an HLR. The training dataset (including tolerance, confidence metrics, or other performance criteria) specifies what the component should do. It is also an HLR that refines the top-level HLR of the component.

The test dataset is held back from original data and serves as source of truth for checking the result of the training process. The test dataset (along with acceptance tolerance, confidence metric, etc.) is used to evaluate the inference model produced by ML training and corresponds to the requirements-based test cases required by DO-178C for verification of HLR.

The result of the ML training process is the inference model that specifies how the ML-based component will compute its outputs. This trained model is the Low Level Requirements for the ML-based component in accordance with DO-178C. Traditional software development methods can be applied to produce Source Code as usual from these LLR. Model-based development (MBD) code generation methods can also be used to produce source code from these LLR.

In summary:

- The training dataset and ODD are HLR for the ML-based component. They define what the component should do and are developed from the HLR originally allocated to the ML-based component. Therefore, this approach involves multiple levels of HLR.
- The test dataset provides the requirements-based test cases for the ML-based component HLR
- The inference model produced by the ML training process is the LLR that describe how to implement the source code for the ML-based component. These LLR define how the component computes its outputs.

With this alignment of terms, development of an ML-based component can be considered in two parts:

- ML-related activities that transform requirements into an inference model. These activities will refine a subset of the item HLR into additional HLRs (ODD and datasets) and use these to train a model that can be treated as LLR for software development.
- Traditional activities that transform the model into embedded software that will be deployed on the aircraft. These activities translate the LLR produced by the ML training process into Source Code and, finally, Executable Object Code (EOC).

The assurance activities and objectives can be broken down similarly and aligned with the purpose of software verification as described in DO-178C:

- Show that the inference model produced by the ML training process is correct, meaning that the LLR satisfy the HLR. We must also perform activities to provide confidence in the absence of unintended function in the model, which is the purpose of trace data from LLR to HLR. Some new or different activities will be needed to accomplish this, as discussed in the next section.

- Show that the embedded software is correct, meaning that it satisfies the LLR corresponding to the inference model and provides confidence in the absence of unintended functionality in the software. This can be accomplished using the same activities as for traditional software.

IV. AFFECTED ACTIVITIES AND OBJECTIVES

Several traditional activities and objectives in DO-178C may be impacted in ML-based component development. Some objectives may need to be interpreted in a different way to account for differences in ML-based component development, but they still provide the same type of assurance. However, other objectives and activities may no longer contribute to assurance as expected and will need to be replaced, updated, or augmented. This is because, unlike traditional software (or hardware), the behavior of a trained NN model cannot be understood by decomposition or examination of its individual parts. The following paragraphs describe objectives that will be impacted in some way.

A. High-level requirements are developed

HLR defining the ML-based behavior of the component will need to be developed into one or more ODD and associated training and test datasets. They provide the necessary inputs to the ML training process. These are new types of HLR specific to the ML development process, but they should be considered as HLR subject to the associated verification objectives.

Note that the ODD concept is not unique to ML. The ODD is equivalent to normal range inputs and conditions that are used to define normal range test cases in DO-178C.

B. Software architecture is developed

The ML-based component (or NN) is a component in the software architecture. This is part of the decomposition of the functionality specified in System Requirements allocated to the software item. In many cases, a NN will rely on pre- and post-processing performed by traditional software components to perform functions such as input monitoring (to detect inputs that are outside the ODD), normalization, or output selection. These pre- and post-processing functions should also be specified in the software architecture.

C. Low-level requirements are developed

Low Level Requirements are software requirements from which Source Code can be directly implemented without further information. The inference model produced by the ML training process should therefore be considered LLR for this component in the software architecture. It provides the information needed to create the source code that implements the inference model. Since the training process output (the inference model) is verified by subsequent activities, there should be no need to qualify any tools used in the training process.

An additional category of lifecycle data to be developed as part of the ML training process defines the model structure and constrains how training will be performed. This includes parameters such as the number of network layers, number of nodes/neurons in each layer, the computation performed by the neurons (summation, convolution, etc.), and the activation function to be used. Although this information is an input to the

training process, it is also part of the detailed description of the inference model necessary for implementation of source code. These parameters should therefore be considered part of the software design description and are, therefore, part of the LLR.

The ML training process is analogous to the design process performed by human engineers in traditional software development, whereby the HLR are transformed into the software design, including the LLR. The difference is that the ML training process is opaque, untraceable, and does not allow inspection of why the details of the LLR are what they are.

D. High-level requirements comply with system requirements

ODD and training/test datasets are HLR that should be reviewed to show that they comply with higher level HLR or System Requirements. This may be shown by demonstrating that the datasets comply with sampling or coverage requirements that have been allocated to the component. If sampling or coverage characteristics of the datasets are developed or refined as part of the training process, they should be considered *derived requirements* (i.e., they do not trace to higher level requirements) and must be shared with the system and safety assessment processes.

There are many ways that datasets may be collected or created. Data may be collected and classified during operation of a reference system. Data may be generated by a mathematical model describing the desired behavior. Data may consist of images that are labeled with information about location and classification of objects in the image. Review or other verification activity will be needed to show that the training and test datasets comply with system requirements or other higher level HLR.

E. High-level requirements are accurate and consistent

It is important to show that training and test datasets are representative of the ODD and provide sufficient coverage of ODD to satisfy verification objectives. Furthermore, the training and test datasets must be shown to be disjoint (no data point is found in both datasets). Documentation and justification of these new features (representativeness, coverage, and disjoint) can be considered ML-specific aspects of this objective.

F. High-level requirements are traceable to system requirements

Traceability is still possible and valuable at this level in the development process. Training and test datasets should be traceable to the ODD. The ODD should in turn be traceable to HLR or System Requirements describing the function of the item containing the ML-based component. If the ODD is not actually traceable to other HLRs (for example, if it was defined based on the training dataset) then it should be treated as a derived requirement and shared with the system and safety assessment processes.

G. Low-level requirements comply with high-level requirements

Satisfaction of this objective is primarily based on statistical performance over the test dataset rather than the individual test cases used in traditional software. Acceptance criteria will be based on statistical measures such as confidence bounds for

regression models or confusion matrix values (false positive, false negative, etc.) for classification models. These criteria may be provided as HLR that must be complied with, or they may be derived requirements developed during training that must be shared with the system and safety assessment processes.

H. Low-level requirements are accurate and consistent

In some implementations, the ML model may be decomposed into separately trained models to cover different parts of the ODD. Alternatively, different layers of an ML model may be implemented in separate items for computational efficiency. In such cases of decomposition, the LLRs corresponding to the individual inference model should be shown to be accurate and consistent, not omitting, duplicating, or changing any requirements.

I. Low-level requirements are traceable to high-level requirements

None of the design elements that appear in the inference model (weights, layers, activation functions of the NN) will be traceable to anything in the HLR. To understand how to handle this objective, it is important to see that traceability is a means, not an end in itself.

DO-248C states that the Trace Data required by DO-178C should be developed to enable the verification of the complete implementation of the requirements, to give visibility to derived requirements, and to support the requirements-based test coverage analysis. DO-248C also provides rationale for this activity, stating that traceability between HLR, LLR, and Source Code provides some assurance that no unspecified functionality exists. In this case, showing that all LLR trace to HLR is a means to show that all elements in the software design are there for a reason and do not implement any unintended function (i.e., something not found in the HLR).

Since we can no longer rely on this method for detection of unintended function in the inference model, this objective must be replaced by some new activity to provide comparable confidence (see Sec. IV.M).

J. Executable Object Code is robust with high-level requirements

The ODD for the ML-based component defines the bounds for normal range testing, analogous to traditional software. Response outside of ODD is analogous to abnormal range testing, which is the focus of this objective for showing that the Executable Object Code (EOC) is robust. It is a design decision to determine how abnormal range testing and evaluation is accomplished. This could be done as part of the NN training and verification by training over an expanded dataset or by adding an input monitor that detects inputs outside of the ODD and triggers appropriate response using traditional software. In either case, additional test cases with associated performance requirements must be created to demonstrate correct response to inputs that are outside the ODD, just as for traditional software.

Showing that EOC is robust with LLR is unchanged from traditional software since the development of the Source Code and EOC from LLR follows a traditional development process.

K. *Test coverage of high-level requirements is achieved*

Test coverage of ML-related HLR is achieved by executing all the points specified in the test dataset and computing the corresponding performance metrics.

L. *Test coverage of software structure is achieved*

In principle, requirements-based testing should result in complete exercise (or coverage) of the software structure. If this is not the case, it may mean that the testing was not sufficiently thorough, the requirements were not complete, or there is extraneous or dead code in the software item. This objective requires action to reconcile any gaps in software coverage according to these possibilities.

According to DO-248C, one of the purposes of structural coverage analysis is to provide a means to support demonstration of the absence of unintended function. Unintended function could be inserted during development of LLR (inference model) from the HLR (training data) or during development of the Source Code and EOC from the LLR. Structural coverage analysis is useful for detecting dead code or unintended function that may have been inserted during the (traditional) process of developing source and EOC from the LLR. However, structural coverage analysis cannot detect unintended function in the inference model that may arise from inadequate or erroneous training [7]. New activities must be performed in addition to this objective to provide comparable assurance (see Sec. IV.M).

M. *New objective: Assurance is obtained that unintended function has been eliminated*

As discussed above, the traditional activities of tracing all LLR to HLR and structural coverage analysis are performed to provide confidence in the absence of unintended function. The purpose is to determine if there is any part of the implementation that has not been evaluated by the verification process (requirements-based test cases) that could still be triggered during operation and result in incorrect or unsafe behavior. These activities are means to an end (detecting unintended function), but they are not the end in themselves. They are methods that work well for the traditional software, but for ML-based components they are unable to satisfy their underlying purpose of detecting and preventing unintended function that could impact safety.

In an ML-based component, unintended function could occur due to inadequate training or bad design choices for the NN structural parameters (e.g., number of layers and neurons). This manifests itself as NN outputs produced in response to previously unseen inputs (data points) that violate the intended function of the item or result in unsafe behavior.

Generalization is a term used in ML methodology to characterize model performance on unseen data. *Stability* and *robustness* are related measures of model performance that provide bounds on model outputs in response to input deviations around a known data point. They can all be viewed as different ways of trying to detect unintended function.

To understand generalization, we consider some ways that unintended behavior may arise in the training process. The ML model is developed using a training dataset and its performance

is evaluated on a separate test dataset to estimate its generalization ability. Overfitting occurs when a model learns the training data too well, resulting in good performance on training data but poor performance on new data. Underfitting occurs when a model is too simple to capture the underlying patterns in the training dataset, potentially leading to poor performance on both training and new data. The generalization gap is the difference between model performance on the training dataset and its performance on unseen data. A smaller gap indicates better generalization.

The better the training dataset covers and represents the characteristics of the ODD, the smaller the generalization gap will be, and the model will be less likely to include unintended functionality. There are several methods that can be used to measure generalization, detect unintended function, or design the model to minimize unintended functionality.

Statistical methods can be used to establish confidence bounds for inference model outputs being within the required tolerance range. In some cases, this may be no different from the performance requirements that are verified for the ML inference model (LLR comply with HLR). This approach may be more appropriate for lower-criticality applications, or for items trained using large synthetic test datasets (allowing tighter confidence bounds to be achieved).

A number of formal methods tools are currently available for comprehensive analysis of some NNs, depending on complexity, architecture, and activation functions used. The robustness analysis enabled by some of these tools (different from the robustness described in Sec. IV.J) can be used to show that NN outputs lie within a required tolerance range for all inputs in the ODD, providing absolute guarantees rather than statistical characterization. Alternatively, they can be used to detect out-of-tolerance results corresponding to unintended behavior, allowing for subsequent retraining and repair. Use of a formal methods tool to evaluate NN generalization may require qualification if the tool output is not independently verified. This approach will be illustrated in Sec. V.M.

Some NN architectures are able to provide improved robustness and generalization results by construction. For example, the output of a 1-Lipschitz NN changes by no more than the corresponding change in its input. The function is constrained so that its Lipschitz constant (the maximum slope or rate of change) is less than or equal to 1, providing bounds on the response to unseen inputs in the neighborhood of a known data point [9].

Run-Time Assurance (RTA) architectures can be used to prevent unintended functionality in a NN from impacting system safety [10]. A monitor function is used to check the outputs of the NN and switch to a simpler, trusted backup function if a safety violation is detected. RTA decouples performance from safety by allowing the ML inference model to run most of the time but intervenes to maintain safety when unintended behavior is detected.

The approach used in a particular ML-based component will depend on system design, safety needs, and specific details of the ML model (e.g., its size and complexity).

N. Leveling

The objectives and activities of DO-178C are modulated based on their Software Level, which is synonymous with the term Design Assurance Level (DAL) used in other standards. The general principle is that higher levels of rigor (corresponding to more objectives that must be satisfied) are required for higher criticality items. There is no qualitative relationship between DAL the objectives required.

The leveling for the activities and objectives described here should follow the leveling in DO-178C. An observant reader will have noticed that LLR and associated verification objectives are not required for DAL D. This means that the objectives discussed in Sec. IV paragraphs C, G, H, and I would not be required for a ML-component in a DAL D item. Furthermore, the objective for absence of unintended function (M) would not be required since structural coverage analysis is not required for a DAL D item.

New activities used to detect or prevent unintended function in ML inference models have some analogy to the objective for model coverage analysis found in DO-331. Unintended function could be included in a model produced in a model-based development process, but this might not be detected by structural coverage metrics computed over requirements-based test cases generated from that model. Model coverage analysis was added to account for this. Following the same principle, new activities for providing confidence in the absence of unintended function in an ML component should similarly be required for DAL A/B/C.

For items assigned DAL A or B, outputs that could contribute to loss of safety must be extremely improbable or extremely remote. For low complexity NNs, it may be possible demonstrate absence of unintended function by means of very high statistical confidence (and low error rates) using an extremely large test dataset. Low to medium complexity NNs may also be analyzed using current formal methods tools to demonstrate absence of unintended function over the entire ODD. However, for complex NN such as those used for vision or perception applications, it may be difficult or impossible to achieve extremely high accuracy that might be expected for DAL A or B items. Depending on the specific system design and safety requirements, complex perception NNs may require an RTA architecture to be used in DAL A or B applications.

V. EXAMPLE

The impacted DO-178C objectives, and their new interpretation proposed in Sec. IV, have been assessed on the OpenRCL case study. OpenRCL is an ML-based component involved in the computation of the Recommended Cruise Level (RCL) function of an aircraft. Its function is to estimate the required fuel and time to cover the next 500 nautical miles distance at a candidate cruise level. These outputs are used to provide pilots with a recommended altitude for maximum efficiency. OpenRCL takes as inputs the weight of the aircraft, a candidate cruise altitude, the temperature, the wind and the air speed (see Figure 1).

OpenRCL is trained on data from the reference implementation of the RCL function, while offering reduced execution time and memory footprint. The reference

implementation is a first principles optimization based on physical models, which is computationally expensive. Additionally, it uses performance lookup tables that require significant memory on the embedded platform.

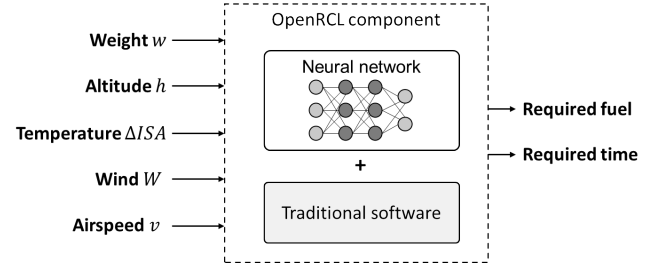


Figure 1. OpenRCL inputs and outputs.

The following paragraphs describe how the development of the OpenRCL component satisfies the objectives and interpretations, corresponding to the definitions in Sec. IV.

A. High-Level requirements are developed

The HLRs that are specific to the ML development process for OpenRCL have been developed:

- **HLR-1:** The RCL in the Cruise flight phase shall be computed using the OpenRCL neural network.
- **HLR-2:** The software shall only compute the fuel and time values when the inputs are valid and fall within the ranges specified in **Table 1** (the function is unavailable).
- **HLR-3:** The region of inputs that are excluded from HLR-2 shall be identified and no value shall be computed.
- **HLR-4:** The OpenRCL neural network shall be trained using the OpenRCL training dataset.
- **HLR-5:** The mean relative error plus one standard deviation of the OpenRCL neural network on both the fuel output and the time output shall not exceed 5% compared with the reference implementation.
- **HLR-6:** The level of confidence that HLR-5 holds shall be at least 99.9%.

Table 1. The input ranges and data resolutions of OpenRCL.

Variable (Unit)	Min	Max	Min res.	Max res.
Weight (lb)	25000	47000	250	500
Altitude (ft)	21000	40000	1000	1000
ΔISA (°C)	-10	20	2	10
Wind (knots)	-250	250	2	2
Airspeed (knots)	160	290	1	2

OpenRCL training dataset: The OpenRCL dataset was developed to cover the input ranges specified in **Table 1**. It was then split into training and test datasets using uniform sampling to guarantee ODD coverage by both datasets. Details on dataset creation and verification are provided in Sec. V.D.

B. Software architecture is developed

The software architecture has been defined (see **Figure 2**). It includes pre- and post-processing functions which consist of min/max normalization and denormalization, respectively, using the values given in Table 1. The architecture also includes a monitor that will check (i) whether the input falls outside of the specified ranges (as per requirement HLR-2) or (ii) whether the input belongs to a region excluded from HLR-2 (as per requirement HLR-3). Such regions correspond to the OpenRCL limitations identified with formal analysis in Sec. V.M. If either (i) or (ii) is satisfied, then OpenRCL does not compute any value.

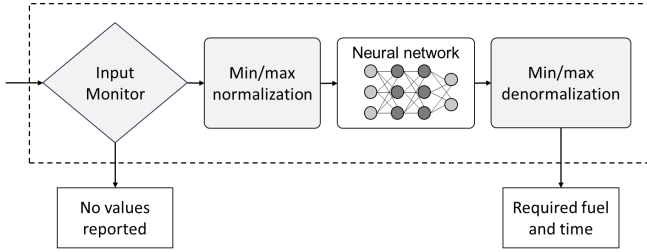


Figure 2. Architecture of the OpenRCL component.

C. Low-level requirements are developed

The design description has been defined, including the hyperparameters used for the ML training process. The Rectified Linear Unit (ReLU) activation functions were selected because they are most amenable to formal analysis. The AdamW training algorithm was chosen for its good balance between faster training and better performance on new data. The Kaiming Uniform Initialization was used to reduce the risk of vanishing or exploding gradient during training with ReLU activation functions. The Mean Squared Error (MSE) function was selected to measure the training loss, as a conventional loss function used in ML regression problems. Several other hyperparameters were selected using hyperparameter tuning with the Optuna tool (e.g., NN structure, learning rate). All the hyperparameters used for training and the NN structure are captured as design data to ensure reproducibility and are summarized in **Table 2**.

Table 2. Training hyperparameters of the OpenRCL neural network.

Parameter	Value
NN structure	5-32-32-2
Activation function	Rectified Linear Unit (ReLU)
Training algorithm	AdamW
Weight initialization	Kaiming Uniform Initialization
Number of epochs	2000
Stopping criteria	The error starts increasing
Learning rate	0.001713
Weight decay	0.000219
Minibatch size	132000

Loss function	Mean Square Error (MSE)
---------------	-------------------------

As per requirement HLR-4, the OpenRCL neural network was trained on the OpenRCL training dataset, and the training environment was documented. Training was performed on an Ubuntu machine equipped with NVIDIA H100 GPU with 80GB memory, using GPyTorch 1.14.2 running on Python 3.10.12. Additionally, the OpenRCL neural network was stored in the ONNX (Open Neural Network Exchange) format, using ONNX 1.19.1 (see visualization in **Figure 3**).

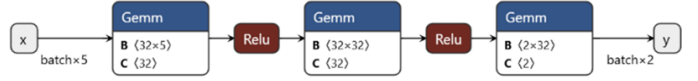


Figure 3. Visualization of the OpenRCL neural network in ONNX.

D. High-level requirements comply with system requirements

The OpenRCL dataset D was created by sampling the input ranges with the resolutions specified in **Table 1**. These resolutions were obtained with sensitivity analysis of the reference implementation to ensure that the sampled data is sufficient to correctly represent the RCL function. The resulting input samples were then labeled by computing the corresponding fuel and time using the reference implementation that is based on a first-principles physics model. This resulted in a set of 123,899,823 labeled data points covering the entire input range.

As a second step, the training dataset D_{train} was obtained by uniformly sampling 80% of the points from the dataset D (its resulting size is 99,123,429 points). The remaining 20% points were used to create a test dataset D_{test} (see Sec. V.G).

E. High-level requirements are accurate and consistent

OpenRCL data was sampled using a uniform grid-based multi-resolution approach that covers the entire range of the ODD, which guarantees data completeness. It was split into two disjoint datasets D_{train} and D_{test} using standard sampling techniques. The sampling distribution is uniform, which ensures that both datasets have the same distribution as the original dataset D .

Completeness of the datasets was checked using an analysis method that splits the input space into subregions called hypercubes. The corners of these hypercubes correspond to the points in the grid from the dataset D . The method then checks, for both D_{train} and D_{test} , whether they contain at least one point in each hypercube, and computes a completeness metric as the ratio between the “covered” hypercubes and the total number of hypercubes. For D_{train} , the metric amounted to 100%, while for D_{test} (due to smaller number of points) the metric initially reached 82%. To ensure completeness of D_{test} , it was then augmented with additional points sampled in the uncovered cubes, while maintaining disjointness with D_{train} .

F. High-level requirements are traceable to system requirements

Generation of the datasets was driven by ODD ranges from **Table 1** which establishes the traceability between ODD and data. The ODD definition (input ranges) was created directly

from system requirements, satisfying traceability to those requirements.

G. Low-level requirements comply with high-level requirements

This section evaluates the OpenRCL compliance with requirements HLR-5 and HLR-6. These requirements specify that the inference model's mean relative error plus standard deviation does not exceed 0.05, with a confidence level of at least 0.999. Prediction accuracy is quantified using the relative error metric, defined as the absolute difference between the model output and the ground truth value, normalized by the ground truth value. Since ML models inherently introduce approximation errors, a statistical assessment is necessary to determine whether the requirements are satisfied with high confidence. Two population-level quantities are examined: MEAN, representing the expected relative error, and SDV, representing the variability of that error. Their empirical counterparts, derived from N samples, are denoted by mean and sdv. A primary challenge in verifying the HLRs is that HLR-5 and HLR-6 refer to the true population values MEAN and SDV, which are unknown. Therefore, the empirical estimators mean and sdv were used as approximations. This is addressed by constructing confidence intervals using generalization bounds that quantify the deviation between empirical and population quantities [11]. For sufficiently large sample sizes, these bounds guarantee that the true MEAN and SDV lie within the empirical estimates plus or minus a specified tolerance at the chosen confidence level.

Table 3 presents the confidence intervals obtained for the two OpenRCL outputs with a sample size of 2.5×10^6 . These intervals specify, with the confidence of 0.999, the ranges in which the true MEAN and SDV values lie. For instance, the empirical mean for the Fuel output is 0.01267. Applying a tolerance of 0.01 yields a confidence interval of [0.01167, 0.01367], indicating that the true MEAN falls within this range with a level of confidence (LOC) of 0.999.

Table 3. Confidence intervals for Fuel and Time.

OUTPUTS	MEAN	SDV	LOC MEAN	LOC SDV
Fuel	[0.01167,0.01367]	[0.00975,0.01175]	0.999	0.999
Time	[0.01913,0.02113]	[0.01543,0.01743]	0.999	0.999

Using the values in **Table 3**, the evaluation of HLR-5 and HLR-6 is performed. Both requirements state that the sum of the mean relative error and the standard deviation remains below 0.05 at a confidence level of at least 0.999. In this assessment, the upper bounds used in the evaluation correspond directly to the upper limits of the confidence intervals in **Table 3**, which represent the worst-case plausible values for the true MEAN and SDV at the 0.999 confidence level. Using these upper-bound values, the resulting sums for the Fuel and Time outputs are 0.02542 and 0.03856, respectively—both well below the 0.05 threshold. This demonstrates that the OpenRCL neural network for fuel and time prediction satisfies requirements HLR-5 and HLR-6.

H. Low-level requirements are accurate and consistent

OpenRCL is implemented as a single item and no decomposition of LLRs between different items is required. Therefore, this part is not applicable to the OpenRCL example.

I. Low-level requirements are traceable to high-level requirements

As discussed in Sec. IV.I, traceability between LLRs and HLRs cannot be performed and does not contribute to the ML assurance process, so this objective is not applicable.

J. Executable Object Code is robust with high-level requirements

OpenRCL is shown to be robust with respect to the HLR by defining and handling abnormal range inputs to the NN – i.e., inputs that are outside of the ODD. The RCL software architecture includes a traditional input monitoring component to handle inputs outside of the ranges specified in Table 1. For these values, no output will be computed in accordance with HLR-2. Testing of the input monitor is in accordance with traditional methods and is out of scope for this paper.

In addition, formal analysis has been carried out over the entire ODD, as described further in Sec. V.M, which aims to prove the absence of unintended behavior of the neural network for any point inside the ODD, including regions and points at the boundaries of the ODD. Any regions on the border of the ODD where proof could not be obtained are also excluded by the input monitor in accordance with HLR-3. This provides further assurance that OpenRCL is robust with HLR.

K. Test coverage of high-level requirements is achieved

Test coverage of HLR-5 and HLR-6 was achieved by executing all the points specified in the test dataset D_{test} and computing the corresponding performance metrics, as described in Sec. V.G.

L. Test coverage of software structure is achieved

Traditional structural coverage analysis will be performed on NN code developed from the LLR but was out of scope for this evaluation. As discussed in Sec. IV.L, structural coverage analysis cannot detect unintended function in the inference model that may arise from inadequate or erroneous training. Therefore, this assurance contribution of the objective is addressed by the objective described next.

M. Assurance is obtained that unintended function has been eliminated

To provide confidence in the absence of unintended function, a formal analysis of the OpenRCL neural network was performed. The goal was to evaluate its behavior on inputs not seen during the training, to ensure the absence of unintended function which could negatively impact safety. For this purpose, an approach referred to as the hypercube-based analysis was applied [12]. In this approach the input space was split into smaller sub-regions where the behaviors of the reference implementation and OpenRCL were compared.

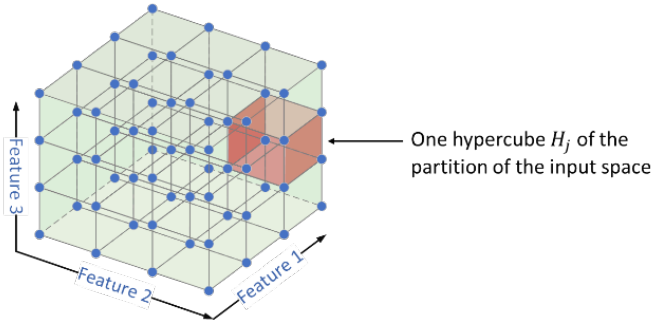


Figure 4. Hypercube partitioning of the OpenRCL input space.

To implement this approach, as illustrated in **Figure 4**, the input space was partitioned into a set of hypercubes, the vertices of which are the data points in the dataset D , obtained by sampling the input ranges with the resolutions specified in **Table 1**. Then, for each hypercube H_j pertaining to this partition, both the range of the reachable outputs by the reference implementation and by OpenRCL were computed (see **Figure 5**):

1. Reachable outputs by the reference implementation, denoted by R^* , were computed using the data points at the vertices of H_j (which are available by construction) and the assumption of monotonicity within the hypercube (based on the sensitivity analysis). Formally, $R^* = [MIN, MAX]$ where MIN is the minimum output value among the vertices, and MAX is the maximum output value;
2. Reachable outputs by the OpenRCL neural network, denoted by R , were over-approximated using formal methods (abstract interpretation). The input hypercube H_j was propagated through the layers of the neural network, using a specialized abstract domain.

Note that this operation was performed identically for the two outputs of the OpenRCL i.e., both the required fuel and time. For simplicity, **Figure 5** illustrates the operation for one output only. Note also that the data used in this operation (i.e., the dataset D) contain both training and test data (D is the union of D_{train} and D_{test}). It is acceptable to use training data in formal analysis because it does not rely on statistical methods, and no assumption is being made on the distribution and/or independence with other datasets. The neural network is neither trained nor (statistically) tested on the dataset D thus there is no data leakage risk.

It was then verified that the reachable outputs by the OpenRCL are contained in the reachable outputs by the reference implementation, with a tolerance τ corresponding to 5% of the maximum output value. Formally, as illustrated in **Figure 5**, it was verified that the following property $\mathcal{P}$ holds:

$$\mathcal{P}: R \subseteq [MIN - \tau, MAX + \tau] \text{ where } \tau = 0.05 \times MAX.$$

This property ensures the absence of unintended function in OpenRCL. It shows that for each hypercube H_j , the possible perturbations of the vertices (points in the hypercube) do not yield large output deviations compared to the reference implementation.

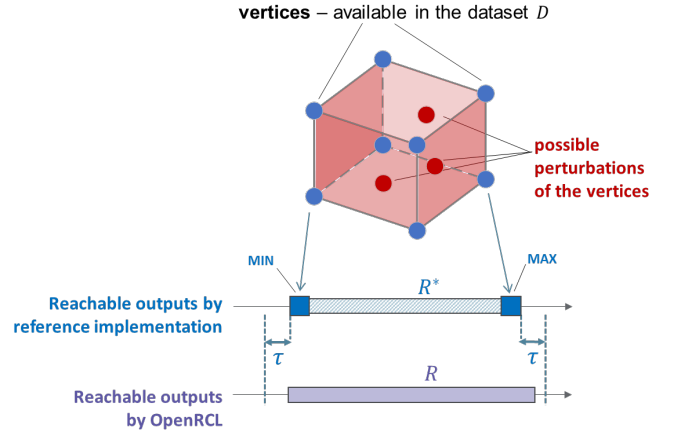


Figure 5. Formal verification of one hypercube H_j .

A first pass of the formal analysis was completed, the results of which are given in **Table 4**. They show that the desired property could be proven valid in 101M hypercubes, among the 107M hypercubes composing the partition. This corresponds to a portion of 93% of the total volume of the input space where the property could be proven valid. In the remaining 7%, the property remains unproven: it can be either a true negative (the property does not hold) or a false negative (due to the over-approximation process with abstract interpretation, the property cannot be proven valid while it holds in reality).

Table 4. Results of the first pass of the OpenRCL formal analysis.

Number of hypercubes analyzed	107M
Hypercubes proven valid	101M (93%)
Hypercubes unproven	6M (7%)
Time elapsed*	11 minutes

*The experiment was run on a Windows machine equipped with an NVIDIA RTX A6000 GPU with 48GB memory.

Following this first pass of the OpenRCL formal analysis, several next steps are possible:

1. A refinement step can be taken to increase the portion of the input space proven valid (i.e., reduce the number of false negatives). Each unproven hypercube can be split into smaller hypercubes by reducing the resolution along each input dimension. Then the verification can be run again on the refined hypercubes. If some hypercubes cannot be proven again, more refinement iterations can be made. This approach was shown to provide successful results in [12].
2. The unproven hypercubes can be analyzed through concrete testing, possibly combined with statistical analysis, to compute tighter bounds on the reachable outputs of the OpenRCL neural network.
3. The property $\mathcal{P}$ can be relaxed by increasing the 5% tolerance τ . This relaxation of the unintended function property can then be reported to the safety team as a derived requirement for the OpenRCL.

4. A verification-driven retraining can be performed by sampling additional data in the unproven hypercubes and retraining the OpenRCL neural network on this new data.
5. The unproven hypercubes can be shared to the safety team and documented as regions to be excluded from the OpenRCL ODD, as required by HLR-3 (see **Figure 6**). Additionally, they can be incorporated into the input monitor.

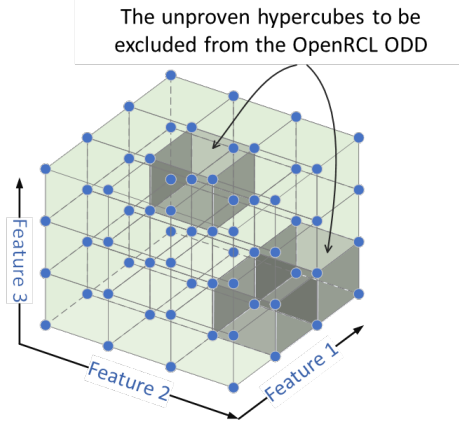


Figure 6. Regions to be excluded from the OpenRCL ODD.

Note that the development of the OpenRCL is still a work in progress and the steps mentioned above have not been explored yet.

N. Leveling

As indicated in Sec. IV.N, the activities and objectives described above are applicable to items assigned DAL A/B/C. While the OpenRCL component itself is DAL D, it is hosted in a DAL B item and must therefore satisfy all of the DAL B objectives.

VI. CONCLUSION

This paper provides an outline of the concerns that should be addressed by an applicant seeking a method of compliance for a software item containing an ML-based component. We have proposed an assurance approach that makes maximal use of the existing terminology, objectives, and activities of DO-178C,

showing how ML might be accommodated with the minimal amount of impact on traditional processes. The principles described here could also be the basis for a new standard or could be incorporated into standards currently under development.

ACKNOWLEDGMENT

The authors are most grateful for helpful discussions with Bob Voros and Kim Wasson.

REFERENCES

- [1] Software Considerations in Airborne Systems and Equipment Certification, RTCA DO-178C, RTCA, Inc., Washington, DC, 2011.
- [2] "Transport Airplane Issues List," Federal Aviation Administration, Dec. 31, 2025. [Online]. Available: https://www.faa.gov/aircraft/air_cert/design_approvals/product_issues_lists/transport_airplane. [Accessed: May 2, 2026]
- [3] EASA Artificial Intelligence Concept Paper Issue 2: Guidance for Level 1&2 machine learning applications. Technical report, EASA, March 2024
- [4] K. S. Wasson and R. Voros, "Deobfuscating Machine Learning Assurance and Approval," in 2024 IEEE/AIAA 43rd Digital Avionics Systems Conference (DASC), San Diego, CA, USA, 2024.
- [5] K. Dmitriev, J. Schumann, and F. Holzapfel, "Toward certification of machine-learning systems for low criticality airborne applications," in 2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC), 2021,
- [6] C.M. Holloway, "Understanding the Overarching Properties," National Aeronautics and Space Administration (NASA), Langley Research Center, Hampton, VA, NASA/TM-2019-220292, 2019.
- [7] Supporting Information for DO-178C and DO-278A, RTCA DO-248C, RTCA, Inc., Washington, DC, 2011.
- [8] D. Cofer, "Unintended behavior in learning-enabled systems: detecting the unknown unknowns," in 2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC), San Antonio, TX, USA, 2021.
- [9] M. Serrurier, F. Mamalet, A. González-Sanz, T. Boissin, J. M. Loubes and E. del Barrio, "Achieving Robustness in Classification Using Optimal Transport With Hinge Regularization," 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Virtual Event, Canada, 2021
- [10] D. Cofer et al., "Flight test of a collision avoidance neural network with run-time assurance," in 2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC), Portsmouth, VA, USA, 2022.
- [11] Abu-Mostafa YS, Magdon-Ismael M, Lin HT. Learning from data. New York: AMLBook; 2012 Mar.
- [12] Clavière A., Kirov D., and Cofer D. How to Verify Generalization Capability of a Neural Network with Formal Methods. In 2025 Symposium on AI Verification (SAIV), Zagreb, Croatia, 2025