Implementation of airborne ML models with semantics preservation

Nicolas Valot[1], Louis Fabre[1], Benjamin Lesage[2], Ammar Mechouche[1], Claire Pagetti[2]
[1] Airbus Helicopters, [2] ONERA

Abstract

Machine Learning (ML) may offer new capabilities in airborne systems. However, as any piece of airborne systems, ML-based systems will be required to guarantee their safe operation. Thus, their development will have to be demonstrated to be compliant with the adequate guidance. So far, the European Union Aviation Safety Agency (EASA) has published a concept paper and an EUROCAE/SAE group is preparing ED-324. Both approaches delineate high-level objectives to confirm the ML model achieves its intended function and maintains training performance in the target environment.

The paper aims to clarify the difference between an ML model and its corresponding unambiguous description, referred to as the Machine Learning Model Description (MLMD). It then refines the essential notion of semantics preservation to ensure the accurate replication of the model. We apply our contributions to several industrial use cases to build and compare several target models.

1 Introduction

Machine Learning (ML) has gained increased consideration even in airborne avionics systems. However, the introduction of ML algorithms in avionic embedded systems challenges the established practices of the development assurance industry. Thus, it has led to the emergence of new development assurance processes, as outlined in the EASA guidance [10] and the yet to be published draft of the ED-324, with publicly available material [14, 24, 23]. Both documents are limited to the design assurance levels related to the least critical failure conditions: Major and Minor.

Guidance approach. These documents address supervised, off-line trained, ML models and promote a W-shaped development life-cycle (outlined in Figure 1). It consists roughly of two main phases: 1. the design of the intended function (first V-cycle), and 2. its replication in the Target Model (second V-cycle). The Target Model (TIM) captures both representation of the implemented ML model, and the target environment, i.e. the hardware and software platform and its configuration used to execute the model. The second V-cycle first implements the Machine Learning Model Description (MLMD) into a TIM, while ensuring the semantics preservation of the off-line trained model. The verification then assesses the correct replication of the TFM by the implementation.
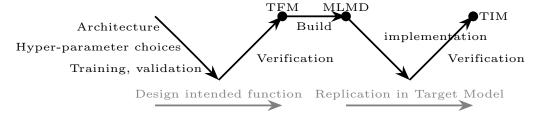

Figure 1: ED-324 W-shape Development Life Cycle

As guidance, the new development assurance processes exhibit some leeway in their interpretation, and they do not aim to be prescriptive of specific tools or methods. This led to a recent, and growing interest in providing techniques to support the certification of ML-based systems across the W-shaped life-cycle [9, 32, 6, 11]. Our approach complements those works with a consistent method to comply with the certification objectives related to the replication phase.

Focus of the paper. This paper focuses on the construction of the MLMD as the *bridge* between the two phases. The Training Framework Model (TFM) results from off-line training and verification. It is expressed using the internal representation of a training framework. By contrast, the MLMD is a non-volatile and semantically-defined ML model, which is independent of training concerns, i.e. the MLMD does not contain learning rate or loss function. To build the MLMD, it is essential to first identify and formalise the TFM behaviour as well as the properties that must be maintained during implementation. The verification at the end of the first V phase aims to determine whether the ML model satisfies

requirements and properties such as stability, generalisation, performance and robustness. Therefore, the implementation is expected to accurately encode the ML model mathematical operations (e.g. convolution or pooling) while preserving all properties fulfilled by the TFM.

Several verification strategies can be employed. First, the applicant replicates all the verification steps performed at the end of the first V phase on the TIM. If any property is not preserved, they must develop a process to correct the implementation. Second, the applicant may choose their preferred training framework (e.g., Keras with PyTorch), fully reverse the TFM-associated code implementation, and re-implement the executable-level semantics identically. This procedure is highly costly and only applicable to a particular version of the training framework, which evolves regularly. We propose an alternative to these two extreme approaches.

Contributions. We propose a definition of semantics preservation supporting the ED-324 objectives and the *Anticipated MOC-SA-01-2* of EASA concept paper [10]. It ensures both the TFM and the TIM satisfy the same properties, as verified by the TFM verification metrics. In effect, the applicant formalises 1. the properties to be kept, 2. the behavioural discrepancies between the TFM and the TIM, and 3. demonstrates that the properties are preserved under certain conditions. We outline a method to assess whether a TIM preserves said properties. We export the TFM in an intermediate format. There are numerous formats available for exchanging ML models between different training or deployment frameworks (Tensorflow-lite, NNEF, StableHLO, OpenVINO, the Open Neural Network eXchange (ONNX)). In this work, we consider ONNX-based MLMD as this format is widely used in the ML domain and embedded domain. Since TFM can be very large, manual production of a MLMD is not sustainable and *exporters* should be considered. We evaluate our approach on industrial use-cases, using state-of-the-art tooling to generate TIM. Our evaluation shows that the resulting models can satisfy the same properties as the TFM, even at reduced numerical precision or on an embedded platform.

In the end, the paper answers the question: can a ML Model be demonstrated as exactly replicated on the target system? Which approaches could be applied for that?

Outline. The paper is organised as follows. Section 2 introduces the description of an ML model, in relation to the ED-324 objectives. It provides the background required to define our proposal to verify the semantics preservation from TFM to TIM. We discuss the application of our method in Section 3, considering how model transformations impact the TIM lifecycle. To support our evaluation, Section 4 introduces the considered TFM case studies and methods to build TIM. The verification of both TFM and TIM is performed in Section 5. Finally, we consider how our approach fits with existing work in Section 6, before concluding in Section 7.

2 How to describe a ML model

ED-324 defines a design assurance process to conceive and develop ML-based systems. One important expectation is to ensure that the TIM reproduces the TFM behaviour and the properties observed at the end of the design, and to facilitate its verification. The approach proposed by the standard is to introduce an intermediate description, the MLMD, between the two V cycles. The purpose of MLMD is to describe ML models in an unambiguous way, so that the implementation can start with a complete specification.

2.1 Understanding which properties to preserve

First, let us briefly outline what is done during the verification of the first V. A ML model is expected to fulfill some requirements and properties such as stability, generalisation, performance, robustness. Practically, several metrics are used to check whether those properties are met, and to identify the conditions under which those properties are fulfilled (e.g. range of input data). Figure 2 highlights this verification by the data scientists at the end of the first V. They consider a set of methods and metrics, and they check that the error between the prediction and the ground truth, for a (combination of) *metric*, fits within acceptable bounds for the test dataset.

Notation 1 *Let us consider a dataset composed of n input samples $\mathcal{D} = \{x_i\}_{i \in [1,n]}$ with $x_i \in \mathbb{R}^p$. Let us denote by $f : \mathbb{R}^p \to \mathbb{R}^m$ the unknown function to be fitted by the ML model. Thus, $f(x_i)$ denotes the ground truth associated to the input sample x_i. Let us denote by $\hat{f}_1 : \mathbb{R}^p \to \mathbb{R}^m$ the function realised by the TFM.*

Example 1 (Metric L_∞) *A very simple metric is given by L_∞: the idea is to check for all $x_i \in \mathcal{D}$ that the prediction of the TFM is close enough to the ground truth, i.e. $L_\infty(f, \hat{f}_1) = \sup_i |f(x_i) - \hat{f}_1(x_i)| \le R_{L_\infty}$ where $R_{L_\infty} \in \mathbb{R}_+$ is an acceptable bound set by the designer.*
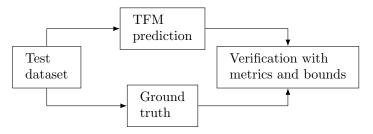
Figure 2: TFM verification with a set of metrics

Example 2 (Metric Bias) *A second widespread metric is the Bias, often used for regression tasks, with* $\mathrm{Bias}(f, \hat{f}_1) = \frac{1}{n} \sum^n \left(\hat{f}_1(x_i) - f(x_i) \right)$. *The associated TFM verification is to check whether* $|\mathrm{Bias}| \leq R_{\mathrm{Bias}}$ *where* $R_{\mathrm{Bias}} \in \mathbb{R}_+$ *is an acceptable bound set by the designer.*

Example 3 (Metric MAE) *Lastly, if* generalisation *is one of the properties, the associated metric is the Mean Absolute Error (MAE), where* $\mathrm{MAE}(f, \hat{f}_1) = \frac{1}{n} \sum^n |\hat{f}_1(x_i) - f(x_i)|$. *Again, the condition to be checked is* $\mathrm{MAE} \leq R_{\mathrm{MAE}}$ *where* $R_{\mathrm{MAE}} \in \mathbb{R}_+$ *is an acceptable bound set by the designer.*

2.2 ED-324 objectives related to MLMD

The second V of the W-shape aims at preserving the properties satisfied by the TFM (see previous section). For this purpose, ED-324 relies on the definition of the MLMD as a bridge between ML model design (TFM) and ML model implementation, that we call the *Target Model* (TIM). The purpose of the MLMD is to ensure that the TIM reproduces the TFM observed behaviour, with the intent that they both satisfy the same properties. This is highlighted in Figure 3.
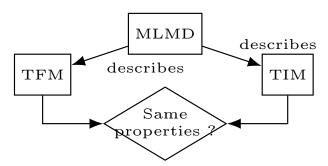

Figure 3: MLMD to guarantee ML Model properties

To achieve this, the standard defines several objectives, and we refine those related to the construction of the MLMD. The ML Model Description (MLMD), as its name suggests, is an artifact that maintains a record of both the ML model design and its final configuration. A ML model is specified as a directed acyclic graph, the nodes of which are the operators and the edge the data-flow between operators. The first refined objective states that these computations are thoroughly captured by the MLMD, and is related to Objective IMP-04 of [10].

Objective 1 (Capturing the elementary functions and composition rules) *The function associated to each operator must be defined in a non ambiguous way, and the rules for combining them, to produce the function carried out by the neural network, must also be provided.*

The second refined objective related to Objective SA-01 and IMP-06 of [10] requires the TIM to *accurately* reproduce the TFM behaviour and this is expressed by *semantics preservation*.

Objective 2 (Semantics preservation) *The MLMD must come with an unambiguous semantics which must be preserved in the TIM.*

Thanks to these objectives, the verification activities done at the end of the design phase do not have to be redone in the TIM. However, those objectives are not prescriptive to allow applicant for choosing their own (internal industrial) process development and as such, they are open to interpretation. Our purpose is to clarify our understanding of their meaning, propose an interpretation with an associated method to achieve them, and bring some formal arguments towards correctness.

2.3 Semantics preservation to maintain ML properties

Let us start with refining the meaning and purpose of those two objectives. Objective 1 concerns the description of the ML model as a directed acyclic graph, capturing the dataflow and operations in the ML model. One natural (and minimal) way to describe such model is a *mathematical representation*. Each operator is a function over $\mathbb{R}$ or $\mathbb{Z}$ defined by mathematical expressions with basic operators (e.g. $+, -, \times, /, \exp$). [31] provides such a formal description for simple neural networks. The notion of semantics preservation of Objective 2 can encompass matching the pure numerical output of a ML model or a more context-dependent definition, such as the classification it produces. Let us complete Notation 1.

Notation 2 *Let us denote by $\hat{f}_2 : \mathbb{R}^p \to \mathbb{R}^m$ the function realised by the TIM.*

Figure 4 represents the different predictions $(f, \hat{f}_1, \hat{f}_2)$ and their satisfaction of a property according to a metric M and a bound R_M. For the sake of simplicity and 2D representativeness, we suppose that $f : \mathbb{R}^p \to \mathbb{R}$.
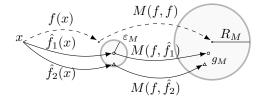

Figure 4: Semantic preservation in the TIM

The acceptable error requirement for a given metric M is depicted by the larger circle of radius R_M in Figure 4. Intuitively, predictions from the TIM are subject to the same constraints as the TFM to preserve the semantics of the ML model. There are some discrepancies between the predictions of the TFM and the TIM [21], due to the different model representations or execution environments. This is highlighted by the small circle centered by the TFM prediction.

Definition 1 (Semantics preservation in the TIM) *The TIM preserves the semantics if it implements MLMD and preserves the verification against the ground truth. In fact, for any metric M and their associated bounds R_M, $M(f, \hat{f}_1)$ is inside the big circle with M computed over $x_i \in \mathcal{D} \implies M(f, \hat{f}_2)$ is also inside the big circle.*

As suggested by Objective IMP-05 in [10], TIM is to be implemented through DO-178/ED-12 [29] development lifecycle for which the metrics M may be included in the High Level Requirements. We propose a method to assess whether the TIM preserves the verified ML model properties by comparing the predictions of the TFM and TIM.

2.4 Proposed approach to ensure semantic preservation

The method relies on the provision by the model designer of all metrics M used during the verification and on an acceptable bound R_M. In this work, we assume that a metric is a function $M : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$. For all metric M, the implementation engineer must derive an error margin ε_M and a positive budget g_M such that:

$$|M(f, \hat{f}_1)| \le R_M - g_M \wedge L_\infty(\hat{f}_1, \hat{f}_2) \le \varepsilon_M \implies |M(f, \hat{f}_2)| \le R_M$$

For some metrics, the constraints over M are to be greater than, i.e. $|M(f, \hat{f}_1)| \ge R_M + g_M \wedge L_\infty(\hat{f}_1, \hat{f}_2) \le \varepsilon_M \implies |M(f, \hat{f}_2)| \ge R_M$. In all cases, if such error margin ε_M can be computed for all metrics budget g_M, then the TIM preserves the semantic of the TFM.

Example 4 (L_∞ metric) *Let us illustrate how to apply the approach when the metric is the L_∞ (see example 1). We have seen that $L_\infty(f, \hat{f}_1) = \sup_i |f(x_i) - \hat{f}_1(x_i)|$. Therefore, $L_\infty(f, \hat{f}_2) = \sup_i |f(x_i) - \hat{f}_2(x_i)| = \sup_i |f(x_i) - \hat{f}_1(x_i) + \hat{f}_1(x_i) - \hat{f}_2(x_i)| \le L_\infty(f, \hat{f}_1) + L_\infty(\hat{f}_1, \hat{f}_2)$. Thus, there exist an acceptable error margin during implementation $\varepsilon_\infty = R_{L_\infty} - L_\infty(f, \hat{f}_1)$ and a formula $g_\infty = \varepsilon_\infty$. Note that in the worst case, $\varepsilon_\infty = 0$ and the implementation must reproduce identically the TFM.*

Example 5 (Regression task and bias metric) *Let us apply the approach on the bias metric (see example 2). We have seen that $\text{Bias}_1 = \frac{1}{n} \sum^n \hat{f}_1(x_i) - f(x_i)$. Therefore, $\text{Bias}_2 = \frac{1}{n} \sum^n \hat{f}_2(x_i) - f(x_i) = \frac{1}{n} \sum^n \hat{f}_2(x_i) - \hat{f}_1(x_i) + \hat{f}_1(x_i) - f(x_i) = \frac{1}{n} \sum^n \hat{f}_2(x_i) - \hat{f}_1(x_i) + \text{Bias}^{(1)}$. This entails $|\text{Bias}_2| \le L_\infty(\hat{f}_1, \hat{f}_2) + |\text{Bias}^{(1)}|$. Thus, there exist an acceptable error margin during implementation $\varepsilon_{\text{Bias}} = R_{\text{Bias}} - L_\infty(f, \hat{f}_1)$ and a formula $g_{\text{Bias}} = \varepsilon_{\text{Bias}}$.*

With the previous examples, we see that for each metric, the implementation engineer must find some formula relating $M(f, \hat{f}_2)$ with $M(f, \hat{f}_1)$. We will see in the next section that this is possible for the usual metrics used by data scientist. The second question is how to set ε_M (or the small circle) to fit the budget g_M if necessary. The MLMD can be given at different levels of abstraction. We refer to this concept as *semantics-level-replication* (or SL-replication). We propose 4 levels SLx of semantics inspired by [13, 19, 30], to address Objective IMP-07 of [10]. **SL0 is the pure mathematical notation** mentioned before. The last level SL3 is met when the predictions of the training framework are strictly equal to the TIM predictions. This corresponds to the bit-accurate-replication (i.e. $\forall x_i \in \mathcal{D}, \hat{f}_1(x_i) = \hat{f}_2(x_i)$). SL$k$ is a complementary specification of SL$k-1$ (it encapsulates SL$k-1$). Thus, the higher k is in the SLk at which TFM and TIM match, the smaller is the ε_M (the small circle in Figure 4) leading to a looser constraint over $M(f, \hat{f}_1) \leq R_M - g_M$ but at the cost of reaching SLk.

SL1 - Machine number representation based level: The ML model is specified using a mathematical description similar to the mathematical notation SL0, with functions defined over machine number representations, e.g. 64-bit floating point (FP64) or 8-bit integer (INT8). The explicit behaviour of operations shall be specified for cases where their expected output in $\mathbb{R}$ is outside the range of the selected representation. Indeed, operations over machine number representations might overflow due to the limited precision of the representation. As an example, the INT8 range is $[-128, 127]$, and $127 + 1$ results in an overflow. We denote by $\mathbb{D}_b$ a b-bit encoded representation of domain $\mathbb{D}$, and $\mathbb{F}_b$ in particular denotes b-bit floating point numbers. The §5 of the IEEE-754 standard [1] supports this level of semantics for operations on numbers in $\mathbb{F}_b$ for $b \in \{64, 32, 16\}$.

SL2 - Operational semantic level: The ML model specification should decompose tensor and matrices expressions into an explicit ordering of all elementary operations on scalar values, and the approximations used for all mathematical operations. Indeed, $\mathbb{F}_b$ data types raise associativity concerns. It may be that for some $(a, b, c) \in \mathbb{F}_b^3$ and $op \in \{+, \times\}$, $(a\ op\ b)\ op\ c \neq a\ op\ (b\ op\ c)$. In addition, mathematical functions $(\exp, \tanh, \sigma...)$ are usually specified by a polynomial, resulting in an ordered sequence of op. Therefore, the order of all op computations is defined. The specification is thus akin to three-address code [2] where each instruction features at most three scalar operands, that is an assignment and a binary operator. Some operation orderings may mitigate the rounding error, as example using pair-wise summation, the Kahan algorithm [33] or the usage of augmented arithmetic described in §9.5 of IEEE-754.

SL3 - Execution model level: The ML model specification maps expression operands and operations onto hardware resources. Expression operands might be assigned in registers or memory. Similarly, operations may be mapped to specialised or extended precision arithmetic units, such as fuse-multiply-add (FMA) for an $(x \times y + z)$ operation. The decision to assign an operand to a register, or an operation to a specific unit might result in operations performed at extended precisions over the SL2 specification, thus impacting the rounding effects. At this level, the rounding policy (to nearest, towards zero, ...) is defined which has an effect on each addition, multiplication in $\mathbb{F}_b$. The environment, and in particular the compilation tool-chain, play a crucial role in a SL3 specification. As an example, register spilling occurs when the compiler has to push intermediate operands into memory, instead of registers. The results of computation will be rounded when pushed to memory, while they would not if the computation remained in extended precision registers. At source level (C language), one can enforce rounding using the *volatile* qualifier on local variables which forces data to be pushed and fetched from memory. The optimising compiler might similarly elect to map, or not, an operation onto the FMA.

3 Applicability of the approach

In this section, we detail the approach proposed in Section 2.4 on different metrics to show that it is always possible to construct ε_M and g_M. We then focus on how to build an MLMD at a *replication-semantics-level* SLx. We choose to rely on the ONNX format, one of the most widely-supported by popular training frameworks in the ML community. This makes ONNX a prime candidate as a MLMD format supporting Objective 1.

3.1 Constructing g_M

Let us recall some ML metrics (§2 in [26]). Our list is not exhaustive meaning that an applicant would have to extend them for any other metrics or any other dimensionality (e.g. if M is not in $\mathbb{R}$).

Regression task main metrics The main metrics are the L_∞, Bias, Mean Absolute Error (MAE), Mean Square Error (MSE), Variance, Explained variance score (EVS), Coefficient of determination (R^2), Mean Absolute Percentage Error (MAPE) metrics. The case of L_∞ (resp. Bias) has been shown in Example 4 (resp. 5). The others are shown in Figure 5, the equations are detailed in Section 8.

Classification and Decision task metrics We can extend this method to classification tasks considering that $\hat{f}(x_i)$ are the logits, i.e. the numerical values associated with the classes, and preceding the *argmax* function which selects the

Metric	TFM threshold	g_M
MAE	$\leq R_M - g_M$	$\varepsilon_{\mathrm{MAE}}$
MSE	$\leq R_M - g_M$	$\varepsilon_{\mathrm{MSE}}^2 + 2\varepsilon_{\mathrm{MSE}}|\mathrm{Bias}^{(1)}|$
Var	$\leq R_M - g_M$	$\varepsilon_{\mathrm{Var}}^2 + 2\varepsilon_{\mathrm{Var}}|\mathrm{Bias}^{(1)}|$
MAPE	$\leq R_M - g_M$	$\varepsilon_{\mathrm{MAPE}}(1 + \mathrm{MAPE}^{(1)})$
EVS	$\geq R_M + g_M$	$\dfrac{\varepsilon_{\mathrm{EVS}}^2 + 2\varepsilon_{\mathrm{EVS}}|\mathrm{Bias}^{(1)}|}{\mathrm{Var}(f(x))}$
R^2	$\geq R_M + g_M$	$\dfrac{\varepsilon_{R^2}^2 + 2\varepsilon_{R^2}|\mathrm{Bias}^{(1)}|}{\mathrm{Var}(f(x))}$

Figure 5: Formula g_M for the main regression tasks metrics

highest scoring class. The metric for classification is Top-1 accuracy score, which is the portion of the number of True Positive and True Negative in the number of samples: $\mathrm{Top\text{-}1}(f, \hat{f}_2) = \frac{1}{n}\sum_i \left(\mathrm{argmax}(\hat{f}_2(x_i)) = cid(f(x_i))\right) \geq R_{\mathrm{Top\text{-}1}}$, where cid is the identifier of the ground truth class.

$$\mathrm{Top\text{-}1}(f, \hat{f}_1) \geq R_{\mathrm{Top\text{-}1}} + g_{\mathrm{Top\text{-}1}}$$
$$g_{\mathrm{Top\text{-}1}} = \mathrm{Top\text{-}1}(f, \hat{f}_1) - \min(\mathrm{Top\text{-}1}(f, \hat{f}_2))$$
$$g_{\mathrm{Top\text{-}1}} = \mathrm{Top\text{-}1}(f, \hat{f}_1) - \min(\mathrm{Top\text{-}1}(f, \hat{f}_1 \pm \varepsilon_{\mathrm{Top\text{-}1}}))$$

To each value of $\varepsilon_{\mathrm{Top\text{-}1}}$ corresponds a value of Top-1 and vice versa. To find the maximum $\varepsilon_{\mathrm{Top\text{-}1}}$ to get a given minimal Top-1 score (and $g_{\mathrm{Top\text{-}1}}$), we need to sample values of $\varepsilon_{\mathrm{Top\text{-}1}}$ and compute the Top-1 scores. This builds a Lookup Table (LUT). We can invert the relationship by interpolating the LUT: $\varepsilon_{\mathrm{Top\text{-}1}} = interp(LUT(\mathrm{Top\text{-}1}))$. More generally, the Top-N score is the condition that $cid(f(x)) \in \mathrm{argmax}_N(\hat{f}_2(x))$, where argmax_N returns the N highest $\hat{f}_2(x)$.

Object detection task metrics Object detection models combine regression and classification, computing respectively the coordinates of a bounding box and its class. The commonly used object detector property is a combination of precision and recall summarized by the AP (Average Precision) indicator, which is based on True positive, False positive, and False negative detections [27]. The Intersection over Union (IoU) is the Jaccard index metric related to the predicted bounding box and the ground truth bounding box. Let a be the sample input, and a detection bounding box $\hat{f}_2(a)$ in the TIM. The False Negative counter is incremented for each bounding box in $f(a)$ with no associated prediction $\hat{f}_2(a)$. If there is a bounding box in $f(a)$ such that $\mathrm{IoU}_{(2)} > threshold$ (threshold is usually chosen above 50%) and $cid(\hat{f}_2(a))$ is identical to $cid(f(a))$, the True Positive counter is incremented. Otherwise, the False Positive counter is incremented.

In the figure 6, the bottom right rectangle is the ground truth bounding box $f(a)$, the top left rectangle is the TFM prediction $\hat{f}_1(a)$, and the dashed rectangle is the prediction in the TIM $\hat{f}_2(a)$ determined by the $\pm\varepsilon_{\mathrm{IoU}}$ added to the $\hat{f}_1(a)$ corners coordinates, which corresponds to the minimal IoU. The IoU is defined by the intersection (in purple) of surfaces over their union (combining purple and yellow).
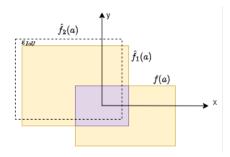

Figure 6: $\min \mathrm{IoU}\ metric$

Let $\{x, y\}$ the coordinates of a ground truth bounding box corner, $\{x_1, y_1\}$ the corresponding TFM prediction, and

$\{x_2 = x_1 \pm \varepsilon_{\text{IoU}}, y_2 = y_1 \pm \varepsilon_{\text{IoU}}\}$ the corresponding TIM prediction, we can bound $\text{IoU}^{(2)} \geq \text{IoU}^{(2)}_{min}$ [1]:

$$
\text{IoU}^{(2)}_{min} = \begin{cases} \frac{(x_1 - \varepsilon_{\text{IoU}})(y_1 - \varepsilon_{\text{IoU}})}{xy} & \text{if } x_1 \leq x, y_1 \leq y \\ \frac{xy}{(x_1 + \varepsilon_{\text{IoU}})(y_1 + \varepsilon_{\text{IoU}})} & \text{if } x_1 \geq x, y_1 \geq y \\ \frac{1}{\frac{x_1 + \varepsilon_{\text{IoU}}}{x} + \frac{y}{y_1 - \varepsilon_{\text{IoU}}} - 1} & \text{if } x_1 \geq x, y_1 \leq y \\ \frac{1}{\frac{y_1 + \varepsilon_{\text{IoU}}}{y} + \frac{x}{x_1 - \varepsilon_{\text{IoU}}} - 1} & \text{if } x_1 \leq x, y_1 \geq y \end{cases}
$$

$$
\text{IoU}^{(2)} = \text{IoU}^{(2)}_{min} \text{ when } \begin{cases} x_2 = x_1 + sign(x_1 - x)\varepsilon_{\text{IoU}} \\ y_2 = y_1 + sign(y_1 - y)\varepsilon_{\text{IoU}} \end{cases}
$$

The $\text{AP}_{\text{IoU} \geq t}$ metric considers AP when $\text{IoU} \geq t$ %.

$$
\text{AP}_{\text{IoU} \geq t}(f, \hat{f}_2) \geq R_{\text{AP}}
$$
$$
g_{\text{AP}} = \text{AP}_{\text{IoU} \geq t}(f, \hat{f}_1) - \text{AP}_{\text{IoU}^{(2)}_{min} \geq t}(f, \hat{f}_2)
$$
$$
\text{AP}(f, \hat{f}_1) \geq R_{\text{AP}} + g_{\text{AP}}
$$

To find the greater ε for a desired minimum AP, one can, similarly to classification tasks, compute $\text{AP}_{\text{IoU}^{(2)}_{min} \geq t}$ for a set of ε_{IoU} in a LUT and find $\varepsilon_{\text{IoU}} = interp(LUT(\text{AP}))$.

3.2   ONNX to support MLMD

ONNX is an open source framework specialized in the inference phase. It relies on three components: ONNX core to describe an ML model, ONNX Runtime (ORT) to execute ONNX models, and ONNX script to transform ML model descriptions. Let us briefly review the ONNX core.

ONNX [3] standardizes in particular linear algebra operators and an Intermediate Representation (IR) to specify an ML model structure. The schematic and simplified representation of the ONNX IR is shown in the UML model of Figure 7. A Model comes with its associated Graph which contains a topologically-ordered set of Nodes, and a set of Initializer values. A Node defines an $Operator_{f,v}$ (op_type), a $domain$, a unique $name$ identifier, and some attributes to particularize its $Operator_{f,v}$. $Operator_{f,v}$ uniquely identifies a $function\ f$ with its version v.

The Graph structure can be constructed through the Node edges which symbolically reference Tensors. Two Nodes are connected if they reference the same Tensor name, the source as an output and the target as an input. The ONNX semantics states that a Node processes its $input$ Tensors and produces $output$ Tensors according to its $Operator_{f,v}$ semantics.

Initializer values represent constant data (trained parameters : weights and biases), and are referenced by Node inputs. Those initializer Tensors are identified by a unique name, an array of dimensions ($dims$), a $data\_type$. A Model also refers to Operator sets ($opset\_import$) which identify a domain name (e.g. $ai.onnx$ is sufficient for neural networks) and a set of $Operator_{f,v}$. A Model also identifies its ONNX file encoding format version ($ir\_version$).

ONNX specifies a mathematical semantics at SL0, compliant more or less with Objective 1. However, the definition of core $Operator_{f,v}$ functions is sometimes lacking critical details. The format also falls short of SL1: while it captures the data types of Tensors and Nodes, it fails to define the expected behaviour of operations outside the range of the selected representations. This is considered outside the scope of this paper. A working group SONNX [2] has initiated the definition of a safety related profile which will refine operators' formal semantics.

3.3   MLMD *Build* methods

We illustrate in Figure 8 the different layers involved in deriving a TIM. The graph vertices are ML model representation and environments, and the edges are SL transformations. On the top level, the designer selects an ML model (e.g. YoLo) that they know to work well for the task they consider (e.g. object detection). Such models are defined in the literature using mathematical operators (SL0).

Then, the applicant selects their favourite training framework, e.g. Keras or PyTorch (respectively on the left and right hand side of the Figure) or they may even reuse a pre-trained model. The training associated activities occur within the training framework, which should capture sufficient SL2+ properties to execute the model. Accounting for the training environment (framework, backend, runtime, etc.), a TFM is semantically defined at SL3.

---

[1] Detailed equations are in Appendix 8.2.

[2] https://github.com/ericjenn/working-groups/tree/ericjenn-srpwg-wg1/safety-related-profile
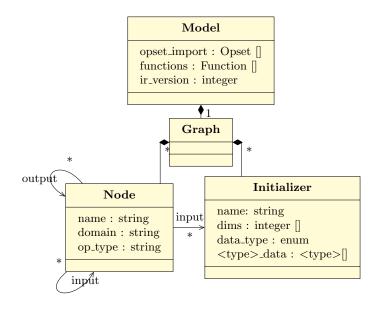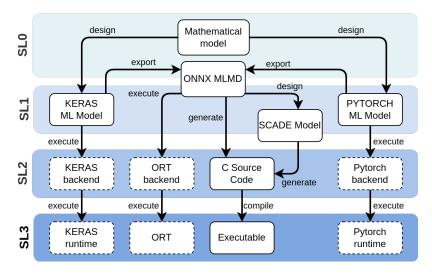
Figure 7: ONNX simplified UML model

Figure 8: MLMD *Build* and Implementation

The MLMD is then constructed from the TFM. As stated in the introduction, the applicant can fully reverse engineer the training framework to try and replicate the SL3 TFM , or try and replicate the exact training environment in the TIM. However, this training environment is hardy a good fit for deployment into resource-constrained, safety-critical environment. ONNX models can also be executed using ORT, which would lead to another SL3 branch. But what we propose instead is to export an ONNX MLMD, and to use code generators from the ONNX description.

Due to the size of ML models, manually deriving the ONNX description or its implementation would be both error-prone and time-consuming. Thus, we rely on tools available in the TFM and ONNX eco-system to build such MLMD from various training frameworks. To build an ONNX model, any exporter tool performs design choices to select ONNX operators, graph, parameters (initializers) matching the semantics of the TFM. Exporters might omit higher SLx properties in the process. Code generators similarly take as an input a model and output a (higher) SL model in the process. They generate code to perform each $Operator_{f,v}$, and the implement the overall data and control flow of the ONNX graph in the generated language and environment. Without a qualified transformer, one with sufficient experience in service, or a review of the original and transformed model, it is unclear that the transformed model is an equivalent representation or a refinement of the original at a higher SL.

(a) `lstm` use case   (b) `linear` use case

Figure 9: Use cases ML model architecture

4 Experimental setup

To evaluate the semantic preservation of ML models (§ 2.3), we consider industrial use cases and their deployment as TIM. We further automate MLMD *Build* and *Replication* methods using tools available in the literature to assess how to derive and verify SL3 TIM, from a TFM . This results in a number of experimental configurations, each denoted as follows:

<div align="center">

`mdl-exp-repr-gen-env`

</div>

The notation identifies the considered use case, and the *Build* method, i.e. technical choices from the TFM (SL1) to the TIM (SL3) as highlighted in Figure 8:

- `mdl` identifies the considered use case and TFM .

- `exp` identifies the exporter from the TFM to ONNX (SL1).

- `repr` identifies the machine representation (SL1).

- `gen` identifies the code generator (SL2).

- `env` identifies the execution environment, that is the compiler, its configuration, and the hardware platform (SL3).

We introduce the considered configurations for each step of the *Build* in the following. Note that a * as configuration values may be used to denote a set of configuration, e.g. `mdl-*-*-*-x86` matches all model `mdl` configurations on the `x86`.

4.1 ML models (`mdl`)

We present two use cases, intended to perform a regression task for helicopter avionics. The TFM for use cases are specified in the KERAS training framework. Both use the `FP32` machine number representation.

4.1.1 `lstm`

The `lstm` model is a Recurrent Neural Network (Figure 9a), used to implement an aircraft weight estimator [25]. It is composed of 3 bidirectional Long Term Short Term Memory (LSTM) operators followed by a dense layer. It takes a 20 input feature vector and has 1 scalar output. The LSTM time frame is made of 16 samples.

4.1.2 `linear`

The `linear` use case was introduced in [16] to compute aircraft loads. Its structure, depicted in Figure 9b, is a Multi Layer Perceptron (MLP) composed of 3 Dense operators, and ReLU activation function.

4.2 ONNX MLMD (`exp`)

Once the TFM has been trained, the MLMD is *built* using the `legacy` KERAS to ONNX exporter. We then apply automated or manual transformations to the exported model using ONNX script. All transformations are SL1-preserving. Indeed, the exported ONNX model cannot be used as is with legacy code generators, which do not support some of the exported Nodes $Operator_{f,v}$ or their parameters. It is allways necessary to tweak the MLMD to be compliant to the subset of the code generator supported ONNX operators.

4.3 Code generators (`gen` and `repr`)

We consider the following open source code generators to generate C code in our evaluation: `onnx2c` [3], et `acetone` [31]. Neither code generator, nor any of the supplemental generators we considered support the LSTM ONNX operator used in the `lstm` model. To replicate the `lstm` use case, we manually designed a *ANSYS Scade 6* [8] model from the ONNX model, and used the `scade` C code generator. The `onnx2c`, `acetone` and `scade` code generator carry over the `FP32` numerical machine representation used in the TFM and in the exported ONNX.

We implemented a mixed precision ONNX runtime which is able to perform inference using various machine representations to assess their impact on semantic preservation. The `ort` generator supports the $orep =$ {`FP64`, `FP32`, `FP16`, `BF16`, `INT16`, `INT14`, `INT12`, `INT10`} data types; `FP`b are defined by IEEE-754, `BF16` is the truncated `FP32`, `INT`b are quantized integer in $\mathbb{Z}_b$. The floating point formats `FP64`, `FP32`, `FP16`, `BF16` respectively offer 52, 24, 11, 8 bits of mantissa precision.

We thus have three distinct sets of `gen` configurations:

- `*-legacy-R-ort-*`: either use-case running on the modified ORT (with `R` $\in orep$);

- `linear-legacy-FP32-C-*`: `linear` state-of-the-art code generators (with `C` $\in$ {`onnx2c`, `acetone`});

- `lstm-legacy-FP32-scade-*`: `lstm` using the `scade` code generator.

4.4 Execution environment (`env`)

We consider two execution environments for our evaluation, a Linux-based Intel server (`x86`), and an NXP T1042 PowerPC (`t1042`). The `t1042` is designed for embedded systems and it does not support the ORT runtime, only configurations that went through a C code generator (`scade`, `onnx2c`, or `acetone` with per use-case restrictions). The generated C code is compiled using conservative compiler options (no optimization). The Windriver® DIAB compiler is used on the `t1042` platform.

We thus explore a subset of the following configurations:

- `*-legacy-FP32-*-t1042`: either use-case running generated C-code on the `t1042` platform;

- `*-legacy-*-ort-x86`: either use-case running ORT on the `x86` platform.

5 Results

The evaluation of the semantics preservation for the considered configurations relies on the approach proposed in Section 3. We first consider for each use-case the properties satisfied by the TFM, and the corresponding budget g_M for various metrics M, to compute the acceptable error ε_M between the TFM and the TIM. We then assess for each experimental configuration whether it fits said requirements, i.e. whether the TIM satisfies the same properties as the TFM .

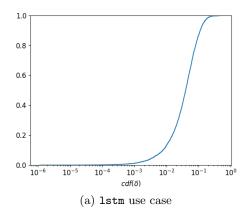## 5.1 Use case `lstm`

5.1.1 TFM verification

The verification dataset is composed of 4288 samples. We first compute the distribution of errors between the TFM and the ground truth, $\hat{f}_1(x_i) - f(x_i)$. Figure 10a presents the corresponding cumulative distribution function of errors. For each error value x, the function provides the portion of the dataset (on the y-axis) with an error less than or equal to x.

The regression metrics for `lstm` are computed in Table 1. For each metric M, the table captures the performance of the TFM ($M^{(1)}$), the acceptable value (R_M) provided by the model designer, and a bound on the error between the TFM and the TIM predictions (ε_M) to ensure semantic preservation. Metric R^2 provides the most stringent requirement, $\varepsilon_M = 0.008$.

5.1.2 TIM verification

We now consider the TIM for the various experimental configurations in comparison to the TFM . We first present the cumulative distribution function of the error $\varepsilon_i = \hat{f}_2(x_i) - \hat{f}_1(x_i)$ between the TFM and each TIM in Figure 11. Each curve represents a different configuration identified by its type (`ort` on `x86`) or platform (`t1042`). Configurations using a reduced numerical precision and thus increased error, `INT`$_b$ or 16-bit floating points, are plotted separately on the right of Figure 11.
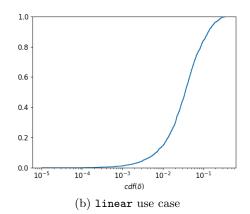
---

[3] https://github.com/kraiskil/onnx2c

(a) `lstm` use case

(b) `linear` use case

Figure 10: Cumulative distribution of errors between the TFM and ground truth $(cdf(\hat{f}_1(x_i) - f(x_i)))$

M	$M^{(1)}$	R_M	ε_M
MAX	0.55	≤ 1.0	≤ 0.44
MAE	0.053	≤ 0.07	≤ 0.017
MSE	0.005	≤ 0.006	≤ 0.014
MAPE	0.079	≤ 0.09	≤ 0.009
R^2	0.841	≥ 0.83	$\leq \mathbf{0.008}$
EVS	0.849	≥ 0.83	≤ 0.013
Var	0.005	≤ 0.01	≤ 0.03
Bias	-0.017	≤ 0.03	≤ 0.012

Table 1: Regression metrics for `lstm`

We see that the errors of the C generated source codes running on the `t1042` environment are very close to that of our ONNX runtime implementation which hints at a similarity between semantics. Unsurprisingly, the error increases (from left to right) as the data type loses precision.

The final assessment is to check for each candidate configuration `lstm-legacy-*-*-*` if it is in the margins of the most constraining R_M of Table 1, i.e. $|\varepsilon_i| < \varepsilon_M \forall i \in [1, n]$. The results are presented in Table 2 capturing for each configuration the maximum observed error $(max(|\varepsilon_i|))$, and its comparison to ε_M. Only the most reduced numerical precisions `BF16`, `INT12`, and `INT10` fail the assertion. All other candidates (in particular the one actually running on the `t1042` environment) replicate the semantics of the TFM .
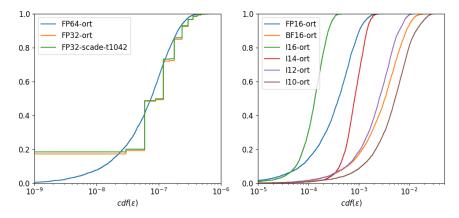

Figure 11: Cumulative distribution function of errors between the TFM and TIM $cdf(\hat{f}_2(x_i) - \hat{f}_1(x_i))$

11

TIM	$\max(|\varepsilon_i|)$	$\leq \varepsilon_M$
`*-*-FP64-ort-x86`	$6e^{-7}$	Y
`*-*-FP32-ort-x86`	$7e^{-7}$	Y
`*-*-FP32-scade-t1042`	$7e^{-7}$	Y
`*-*-FP16-ort-x86`	$3e^{-3}$	Y
`*-*-BF16-ort-x86`	$2e^{-2}$	N
`*-*-INT16-ort-x86`	$4e^{-4}$	Y
`*-*-INT14-ort-x86`	$2e^{-3}$	Y
`*-*-INT12-ort-x86`	$2e^{-2}$	N
`*-*-INT10-ort-x86`	$3e^{-2}$	N

Table 2: Replication metrics for `lstm` with $\varepsilon_M = 0.008$

5.2 Use case `linear`

5.2.1 TFM verification

The verification dataset is composed of 2000 samples. We perform a similar assessment for use-case `linear`. Figure 10b presents the cumulative distribution function of errors between the TFM and the ground truth. The regression metrics and ε_M bounds on the TFM and TIM prediction errors for `linear` are presented in Table 4. The most constraining R_M are EVS and once again R^2 with an $\varepsilon_M = 0.015$.

M	$M^{(1)}$	R_M	ε_M
MAE	0.033	≤ 0.06	≤ 0.026
MSE	0.002	≤ 0.01	≤ 0.088
R^2	0.821	≥ 0.8	$\leq \mathbf{0.015}$
EVS	0.821	≥ 0.8	$\leq \mathbf{0.015}$
Var	0.002	≤ 0.01	≤ 0.088
Bias	-0.0003	≤ 0.03	≤ 0.029

Table 3: Regression metrics for `linear`

5.2.2 TIM verification

Table 4 presents the assessment for each configuration of whether it satisfies to the semantic preservation of the `linear` TFM . Taking into account the tightest constraint in Table 3, both `t1042` TIM are also compliant. Similarly, the higher precision representations FP32, FP16, and INT16 ort TIM are compliant.

TIM	$\max(|\varepsilon_i|)$	$\leq \varepsilon_M$
`*-*-FP32-ort-x86`	$3e^{-7}$	Y
`*-*-FP32-onnx2c-t1042`	$7e^{-7}$	Y
`*-*-FP32-acetone-t1042`	$7e^{-7}$	Y
`*-*-FP16-ort-x86`	$2e^{-3}$	Y
`*-*-BF16-ort-x86`	$1e^{-2}$	N
`*-*-INT16-ort-x86`	$2e^{-3}$	Y
`*-*-INT14-ort-x86`	$1e^{-2}$	N
`*-*-INT12-ort-x86`	$3e^{-2}$	N
`*-*-INT10-ort-x86`	$1e^{-1}$	N

Table 4: Replication metrics for `linear` with $\varepsilon_M = 0.015$

6 Related work

Model exporter tools are designed to translate the training framework semantics into the MLMD semantics. However, varied or repeated translations steps can lead to numerous compatibility and replication issues, reported in the survey [21]. [18] further highlighted the tradeoff between model inference performance and precision when using hardware accelerators.

In the ML community, the ONNX exchange format is one of the most widely-supported by popular training frameworks. Prior works have built MLMD with NNEF [17] and ONNX (Open Neural Network eXchange) [15]. However, these approaches did not explore how to export a TFM, and how to verify its compliance to ED-324 objectives. [15] did consider bit-accurate replication which proved costly considering the number of moving parts involved in the execution of the TFM.

Numerous frameworks can generate C, C++ or CUDA code compatible with airborne systems development life-cycle out of a MLMD [31, 5, 28]. Indeed, in line with legacy guidance DO-178/ED-12, producing source code from the MLMD is a certification-friendly approach. In this work, we rely on implementations produced by C/C++ code generators (`acetone`, `onnx2c`, `scade`) for selected target environment (`x86`, `t1042`). We have chosen code generator methods which are widely adopted in the field of safety critical avionics. The target environment in any case will generate different results than the TFM due to computation roundings studied in [12] and [4], and activation functions algorithm approximations.

7 Conclusion

In legacy avionics development, industrialization and certification require that verification of requirements or properties is performed in the TIM, unless an argument is provided to justify that verification which were performed on a different execution environment are still valid in the TIM. One argument could be a bit-accurate replication between TFM and the TIM which is a quite hard constraint on the development. To avoid this constraint, we propose a ML model semantics preservation definition and a supporting verification methodology, based on state-of-the-art ML metrics. This method relies on two independent processes: 1. the verification of the TFM metrics *including budget* g_M, and using a test set including the ground truth, 2. the verification that the predictions of the TIM fit into *error margin* ε_M using a chosen set of TFM predictions. This second process does not require any knowledge of the metrics, nor the ground truth. It takes as input the MLMD including ε_M. Its objective is to demonstrate an upper bound of the cumulated rounding and approximations errors of the algorithm. In this paper we used the test set input to sample the TFM for the sake of simplicity, but future research might elaborate a sound proof of ε_M threshold. The probability to exceed ε_M threshold can be bounded: $P(\varepsilon > \varepsilon_M) < P_M$ where P_M is a threshold included in MLMD with ε_M. This might require a specific input space sampling. The Objective IMP-08 of the concept paper [10] is limiting verification to the test set which might not be sufficient nor appropriate for implementation verification.

We further decomposed the level of details captured by ML model representations into semantic levels, related to these margins. We considered how existing tools [31, 20] could support the definition of (semi-)automated methods to *Build* a TIM and replicate a TFM, and we showed that they can indeed preserve the properties of industrial ML use cases [16, 7, 22] from the TFM into a TIM.

Acknowledgements

References

[1] IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008*, pages 1–70, 2008.

[2] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison Wesley, August 2006.

[3] J. Bai, F. Lu, K. Zhang, et al. ONNX: Open Neural Network Exchange. https://onnx.ai/, 2019.

[4] T. Beuzeville. *Backward error analysis of artificial neural networks with applications to floating-point computations and adversarial attacks.* PhD thesis, Université de Toulouse, 2024.

[5] A. Cerioli, R. Miccini, C. Laroche, T. Piechowiak, L. Pezzarossa, J. Sparsø, and M. Schoeberl. NeuralCasting: A Front-End Compilation Infrastructure for Neural Networks. *2024 11th International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, 2024.

[6] J. M. Christensen, W. Zaeske, J. Beck, S. Friedrich, T. Stefani, A. A. Girija, E. Hoemann, U. Durak, F. Köster, T. Krüger, and S. Hallerbach. Towards Certifiable AI in Aviation: A Framework for Neural Network Assurance Using Advanced Visualization and Safety Nets. *2024 AIAA DATC/IEEE 43rd Digital Avionics Systems Conference (DASC)*, pages 1–9, 2024.

[7] C. del Cistia Gallimard, K. Nikolajevic, F. Beroul, J. Denoulet, B. Granado, and C. Marsala. Direct Load Recognition to Estimate the Damper Load on the H175 Fleet. In *European Rotorcraft Forum*, 2023.

[8] B. Dion, M. Najork, N. Dalmasso, J.-L. Colaço, O. Andrieu, B. Buettner, T. Most, and J. R. de Amaral. Programming Safety-Critical Neural Network Inference Models. In *11th European Congress on Embedded Real Time Software and Systems (ERTS)*, 2022.

[9] K. Dmitriev, J. Schumann, and F. Holzapfel. Towards Design Assurance Level C for Machine-Learning Airborne Applications. In *2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC)*, pages 1–6, 2022.

[10] EASA. Concept Paper: guidance for Level 1 & 2 machine learning applications - Proposed Issue 02, 2024.

[11] Y. Y. Elboher, R. Elsaleh, O. Isac, M. Ducoffe, A. Galametz, G. Povéda, R. Boumazouza, N. Cohen, and G. Katz. Robustness Assessment of a Runway Object Classifier for Safe Aircraft Taxiing. *2024 AIAA DATC/IEEE 43rd Digital Avionics Systems Conference (DASC)*, 2024.

[12] Y. Fang and L. Chen. Statistical rounding error analysis for random matrix computations. 2025.

[13] R. W. Floyd. Assigning meanings to programs. *Mathematical Aspects of Computer Science*, pages 19–32, 1967.

[14] C. Gabreau and et al. EUROCAE WG114 – SAE G34: a joint standardization initiative to support Artificial Intelligence revolution in aeronautics, 2023. Keynote of SafeAI.

[15] C. Gabreau, M.-C. Teulieres, E. Jenn, A. Lemesle, D. P. Butucaru, F. Thiant, L. Fischer, and M. Turki. A study of an ACAS-Xu exact implementation using ED-324/ARP6983. In *12th European Congress on Embedded Real Time Software and Systems (ERTS)*, 2024.

[16] C. D. C. Gallimard, F. Beroul, J. Denoulet, K. Nikolajevic, A. Pinna, B. Granado, and C. Marsala. Harmonic Decomposition to Estimate Periodic Signals using Machine Learning Algorithms: Application to Helicopter Loads. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2022.

[17] A. Gauffriau, I. De Albuquerque Silva, and C. Pagetti. Formal description of ML models for unambiguous implementation. In *12th European Congress on Embedded Real Time Software and Systems (ERTS)*, 2024.

[18] F. Geyer, J. Freitag, T. Schulz, and S. Uhrig. Efficient and mathematically robust operations for certified neural networks inference. In *6th Workshop on Accelerated Machine Learning (AccML) at HiPEAC*, 2024.

[19] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, page 576–580, 1969.

[20] Y. Ikarashi, G. L. Bernstein, A. Reinking, H. Genc, and J. Ragan-Kelley. Exocompilation for productive programming of hardware accelerators. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI 2022, page 703–718, 2022.

[21] P. Jajal, W. Jiang, A. Tewari, E. Kocinare, J. Woo, A. Sarraf, Y.-H. Lu, G. K. Thiruvathukal, and J. C. Davis. Interoperability in Deep Learning: A User Survey and Failure Analysis of ONNX Model Converters. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 1466–1478, 2024.

[22] J. Jouve, C. del Cistia Gallimard, K. Nikolajevic, and H. Morel. Estimation of confidence margins for Direct Load Recognition (DLR) using supervised and unsupervised machine learning. In *Vertical Flight Society*, 2023.

[23] F. Kaakai and et al. Toward a machine learning development lifecycle for product certification and approval in aviation. *SAE International journal of aerospace*, 15, 2022.

[24] F. Kaakai and et al. Data-centric operational design domain characterization for machine learning-based aeronautical products, 2023.

[25] A. Mechouche, A. Rocher, and V. Aubin. Method for training at least one artificial intelligence model for estimating the mass of an aircraft during flight based on utilisation data. US Patent US20240005207A1 App. 18/209,183, 2024, European Patent EP4300053B1.

[26] MLEAP Consortium, EASA Research. Machine Learning Application Approval (MLEAP) Final Report, May 2024.

[27] R. Padilla, W. L. Passos, T. L. B. Dias, S. L. Netto, and E. A. B. da Silva. A comparative analysis of object detection metrics with a companion open-source toolkit. *Electronics*, 10(3), 2021.

[28] F. S. Perotto, A. Fernandes Pires, J.-L. Farges, Y. Bouchebaba, M. Belcaid, E. Bonnafous, C. Pagetti, F. Boniol, X. Pucel, A. Chan-Hon-Tong, S. Herbin, M. Cassaro, and S. Kraiem. Thinking the certification process of embedded ML-based aeronautical components using AIDGE, a French open and sovereign AI platform. In *International Conference on Cognitive Aircraft Systems*, 2024.

[29] RTCA/EUROCAE. DO-178C/ED-12C - Software Considerations in Airborne Systems and Equipment Certification, 2011.

[30] D. A. Schmidt. Denotational semantics: a methodology for language development. william c, 1986.

[31] I. D. A. Silva, T. Carle, A. Gauffriau, and C. Pagetti. ACETONE: predictable programming framework for ML applications in safety-critical systems. In *34th Euromicro Conference on Real-Time Systems, ECRTS 2022, July 5-8, 2022, Modena, Italy*, 2022.

[32] K. S. Wasson and R. Voros. Deobfuscating Machine Learning Assurance and Approval. *2024 AIAA DATC/IEEE 43rd Digital Avionics Systems Conference (DASC)*, pages 1–10, 2024.

[33] K. William. Further remarks on reducing truncation errors. *CACM*, 1965.

8 Appendix

8.1 Regression Tasks

This appendix details how to compute the ML metrics out of ε_M. The following equations are defining the TIM metric as a function of the TFM metric and the ε_M. δ_i represents the error between TFM prediction $\hat{f}_1(x_i)$ and the ground truth $f(x_i)$, i.e. $\delta_i = \hat{f}_1(x_i) - f(x_i)$. ε_i represents the error between the TIM prediction $\hat{f}_2(x_i)$ and the TFM prediction $\hat{f}_1(x_i)$, i.e. $\varepsilon_i = \hat{f}_2(x_i) - \hat{f}_1(x_i)$. The metric $M^{(1)}$ is related to $\hat{f}_1(x)$ and $M^{(2)}$ is related to $\hat{f}_2(x)$.

MAE metric

$$\mathrm{MAE}^{(2)} = \frac{1}{n}\sum^{n}|\hat{f}_2(x_i) - f(x_i)| = \frac{1}{n}\sum^{n}|\varepsilon_i + \delta_i|$$

$$\leq \mathrm{MAE}^{(1)} + \frac{1}{n}\sum^{n}|\varepsilon_i| \leq \mathrm{MAE}^{(1)} + \varepsilon_{\mathrm{MAE}}$$

$$g_{\mathrm{MAE}} = \max(\mathrm{MAE}^{(2)}) - \mathrm{MAE}^{(1)} = \varepsilon_{\mathrm{MAE}}$$

MSE metric

$$\mathrm{MSE}^{(2)} = \frac{1}{n}\sum^{n}(\hat{f}_2(x_i) - f(x_i))^2 = \frac{1}{n}\sum^{n}(\varepsilon_i + \delta_i)^2$$

$$= \frac{1}{n}\sum^{n}\varepsilon_i^2 + \delta_i^2 + 2\varepsilon_i\delta_i = \mathrm{MSE}^{(1)} + \sum^{n}\varepsilon_i^2 + \frac{2}{n}\vec{\varepsilon}.\vec{\delta}$$

$$\vec{\varepsilon}.\vec{\delta} \in \left[-n\varepsilon_{\mathrm{MSE}}|\,\mathrm{Bias}^{(1)}\,|, n\varepsilon_{\mathrm{MSE}}|\,\mathrm{Bias}^{(1)}\,| \right]$$

$$\mathrm{MSE}^{(2)} \leq \mathrm{MSE}^{(1)} + \varepsilon_{\mathrm{MSE}}^2 + 2\varepsilon_{\mathrm{MSE}}|\,\mathrm{Bias}^{(1)}\,|$$

$$g_{\mathrm{MSE}} = \max(\mathrm{MSE}^{(2)}) - \mathrm{MSE}^{(1)} = \varepsilon_{\mathrm{MSE}}^2 + 2\varepsilon_{\mathrm{MSE}}|\,\mathrm{Bias}^{(1)}\,|$$

Variance metric

$$\text{Var}^{(1)} = \text{MSE}^{(1)} - \text{Bias}^2_{(1)}$$

$$\text{Var}^{(2)} = \text{MSE}^{(2)} - \text{Bias}^2_{(2)}$$

$$= \frac{1}{n} \sum^n \varepsilon_i^2 + \delta_i^2 + 2\varepsilon_i \delta_i - (\bar{\varepsilon} + \text{Bias}_{(1)})^2$$

$$= \frac{1}{n} \sum^n \varepsilon_i^2 + \delta_i^2 + 2\varepsilon_i \delta_i - \bar{\varepsilon}^2 - \text{Bias}^2_{(1)} - 2\bar{\varepsilon}\,\text{Bias}_{(1)}$$

$$= \text{MSE}^{(1)} - \text{Bias}^2_{(1)} + \frac{1}{n} \sum^n \varepsilon_i^2 + 2\varepsilon_i \delta_i - \bar{\varepsilon}^2 - 2\bar{\varepsilon}\bar{\delta}$$

$$= \text{Var}^{(1)} + \frac{1}{n} \sum^n \varepsilon_i^2 + 2\varepsilon_i \delta_i - \bar{\varepsilon}^2 - 2\bar{\varepsilon}\bar{\delta}$$

$$\text{Var}^{(2)} \leq \text{Var}^{(1)} + \varepsilon_{\text{Var}}^2 + 2\varepsilon_{\text{Var}} |\text{Bias}^{(1)}|$$

$$g_{\text{Var}} = \max(\text{Var}^{(2)}) - \text{Var}^{(1)} = \varepsilon_{\text{Var}}^2 + 2\varepsilon_{\text{Var}} |\text{Bias}^{(1)}|$$

Explained variance score metric

$$\text{EVS}^{(2)} = 1 - \frac{\text{Var}^{(2)}}{\text{Var}(f(x))}$$

$$\text{EVS}^{(2)} = \text{EVS}^{(1)} - \frac{\frac{1}{n} \sum^n \varepsilon_i^2 + 2\varepsilon_i \delta_i - \bar{\varepsilon}^2 - 2\bar{\varepsilon}\bar{\delta}}{\text{Var}(f(x))}$$

$$\bar{\varepsilon} \xrightarrow[n \to \infty]{} 0,$$

$$\text{EVS}^{(2)} \geq \text{EVS}^{(1)} - \frac{\varepsilon_{\text{EVS}}^2 + 2\varepsilon_{\text{EVS}} |\text{Bias}^{(1)}|}{\text{Var}(f(x))}$$

$$g_{\text{EVS}} = \text{EVS}^{(1)} - \min(\text{EVS}^{(2)}) = \frac{\varepsilon_{\text{EVS}}^2 + 2\varepsilon_{\text{EVS}} |\text{Bias}^{(1)}|}{\text{Var}(f(x))}$$

Coefficient of determination (R2) metric

$$\text{R}^2_{(2)} = 1 - \frac{\sum^n (\hat{f}_2(x) - f(x))^2}{\sum^n (f(x) - \overline{f(x)})^2} = 1 - \frac{\sum^n (\varepsilon_i + \delta_i)^2}{\sum^n (f(x) - \overline{f(x)})^2}$$

$$= \text{R}^2_{(1)} - \frac{\frac{1}{n} \sum^n (\varepsilon_i^2 + 2\varepsilon_i \delta_i)}{\frac{1}{n} \sum^n (f(x) - \overline{f(x)})^2} = \text{R}^2_{(1)} - \frac{\frac{1}{n} \sum^n \varepsilon_i^2 + \frac{2}{n} \vec{\bar{\varepsilon}\bar{\delta}}}{\text{Var}(f(x))}$$

$$\text{R}^2_{(2)} \geq \text{R}^2_{(1)} - \frac{\varepsilon_{\text{R}^2}^2 + 2\varepsilon_{\text{R}^2} |\text{Bias}^{(1)}|}{\text{Var}(f(x))}$$

$$g_{\text{R}^2} = \text{R}^2_{(1)} - \min(\text{R}^2_{(2)}) = \frac{\varepsilon_{\text{R}^2}^2 + 2\varepsilon_{\text{R}^2} |\text{Bias}^{(1)}|}{\text{Var}(f(x))}$$

MAPE metric The MAPE metric is relative to the sample output value. In this case, we redefine ε and δ as follows: $\hat{f}_1(x_i) = (1 + \delta_i)f(x_i)$, similarly: $\delta_i = \frac{\hat{f}1(x_i) - f(x_i)}{f(x_i)}$. $\hat{f}_2(x_i) = (1 + \varepsilon_i)\hat{f}_1(x_i)$, similarly: $\varepsilon_i = \frac{\hat{f}_2(x_i) - \hat{f}_1(x_i)}{\hat{f}_1(x_i)}$. $\hat{f}_2(x_i) = (1 + \varepsilon_i)(1 + \delta_i)f(x_i)$. Let $(1 + \gamma_i) = (1 + \varepsilon_i)(1 + \delta_i)$, then $\hat{f}_2(x_i) = (1 + \gamma_i)f(x_i)$, similarly: $\gamma_i = \frac{\hat{f}_2(x_i) - f(x_i)}{f(x_i)}$.

$$\text{MAPE}^{(2)} = \frac{1}{n} \sum^n \left| \frac{\hat{f}_2(x_i) - f(x_i)}{f(x_i)} \right| = \frac{1}{n} \sum^n |\gamma_i|$$

$$= \frac{1}{n} \sum^n |(1 + \varepsilon_i)(1 + \delta_i) - 1|$$

$$= \frac{1}{n} \sum^n |\varepsilon_i \delta_i + \varepsilon_i + \delta_i|$$

$$\text{MAPE}^2 \leq \text{MAPE}^{(1)} + \frac{1}{n} \sum^n |\varepsilon_i(1 + \delta_i)|$$

$$\leq \text{MAPE}^{(1)} + \varepsilon_{\text{MAPE}} \frac{1}{n} \sum^n |1 + \delta_i|$$

$$\text{MAPE}^{(2)} \leq \text{MAPE}^{(1)} + \varepsilon_{\text{MAPE}}(1 + \text{MAPE}^{(1)})$$

$$g_{\text{MAPE}} = \max(\text{MAPE}^{(2)}) - \text{MAPE}^{(1)} = \varepsilon_{\text{MAPE}}(1 + \text{MAPE}^{(1)})$$

8.2 Object detection task

1-dim IoU Let us define x the ground truth, x_1 the TFM prediction and x_2 the TIM predictions in 1-dim. We are given three intervals: $I_1 = [0, x_1]$, $I = [0, x]$, and $I_2 = [0, x_2]$, where $x_2 \in [x_1 - \varepsilon_{\text{IoU}}, x_1 + \varepsilon_{\text{IoU}}]$, and $\varepsilon_{\text{IoU}} < |x_1 - x|$. We want to find the minimum $\text{IoU}_{(2)} = \text{IoU}(I_2, I)$ as a function of $x, x_1, \varepsilon_{\text{IoU}}$. The intersection is $I_2 \cap I = [0, \min(x_2, x)]$ and the union is $I_2 \cup I = [0, \max(x_2, x)]$. Thus, the IoU is:

$$\text{IoU}_{(2)} = \frac{I_2 \cap I}{I_2 \cup I} = \frac{\min(x_2, x)}{\max(x_2, x)}$$

We are given that $x_2 \in [x_1 - \varepsilon_{\text{IoU}}, x_1 + \varepsilon_{\text{IoU}}]$ and $\varepsilon_{\text{IoU}} < |x_1 - x|$. We consider two cases based on the relationship between x_1 and x.

Case 1: $x_1 < x$: In this case, $|x_1 - x| = x - x_1$, so the condition becomes $\varepsilon_{\text{IoU}} < x - x_1$, which means $x_1 + \varepsilon_{\text{IoU}} < x$. We have then $x_2 \leq x_1 + \varepsilon_{\text{IoU}} < x$. Therefore, $\min(x_2, x) = x_2$ and $\max(x_2, x) = x$ and $\text{IoU}_{(2)} = \frac{x_2}{x}$. Assuming $x_1 - \varepsilon_{\text{IoU}} > 0$, $\min \text{IoU}_{(2)} = \frac{x_1 - \varepsilon_{\text{IoU}}}{x}$.

Case 2: $x_1 > x$: In this case, $|x_1 - x| = x_1 - x$, so the condition becomes $\varepsilon_{\text{IoU}} < x_1 - x$, which means $x < x_1 - \varepsilon_{\text{IoU}}$. We have $x < x_1 - \varepsilon_{\text{IoU}} \leq x_2$. Therefore, $\min(x_2, x) = x$ and $\max(x_2, x) = x_2$ and $\text{IoU}_{(2)} = \frac{x}{x_2}$. Assuming $x_1 + \varepsilon_{\text{IoU}} > 0$, $\min \text{IoU}_{(2)} = \frac{x}{x_1 + \varepsilon_{\text{IoU}}}$.

Combining both cases, the minimum $\text{IoU}_{(2)}$ as a function of $x, x_1, \varepsilon_{\text{IoU}}$ is:

$$\min \text{IoU}_{(2)} = \begin{cases} \frac{x_1 - \varepsilon_{\text{IoU}}}{x} & \text{if } x_1 < x \\ \frac{x}{x_1 + \varepsilon_{\text{IoU}}} & \text{if } x_1 > x \end{cases}$$

2-dim IoU We can extend this principle to 2-dim rectangular bounding boxes. In the figure 6, we have chosen origin inside the intersection and we consider the IoU for $x > 0, y > 0$, to simplify the symbolic expressions.

Let's consider two bounding boxes, B_1 and B. Let B_1 be defined by the intervals $I_{x_1} = [0, x_1]$ and $I_{y_1} = [0, y_1]$ along the x and y dimensions, respectively. Let B be defined by the intervals $I_x = [0, x]$ and $I_y = [0, y]$. Let B_2 be defined by the intervals $I_{x_2} = [0, x_2]$ and $I_{y_2} = [0, y_2]$, where $x_2 \in [x_1 - \varepsilon_{\text{IoU}}, x_1 + \varepsilon_{\text{IoU}}]$ and $y_2 \in [y_1 - \varepsilon_{\text{IoU}}, y_1 + \varepsilon_{\text{IoU}}]$.

We also set the conditions $\varepsilon_{\text{IoU}} < |x_1 - x|$ and $\varepsilon_{\text{IoU}} < |y_1 - y|$. The IoU of two bounding boxes in 2D is the ratio of the area of their intersection to the area of their union:

$$\text{IoU}(B_2, B) = \text{IoU}_{(2)} = \frac{\text{Area}(B_2 \cap B)}{\text{Area}(B_2 \cup B)}$$

$$\text{Area}(B_2 \cup B) = \text{Area}(B2) + \text{Area}(B) - \text{Area}(B_2 \cap B)$$

$$\text{Area}(B_2 \cap B) = \min(x_2, x) \min(y_2, y)$$

$$\text{IoU}_{(2)} = \frac{\min(x_2, x) \min(y_2, y)}{x_2 y_2 + xy - \min(x_2, x) \min(y_2, y)}$$

$$\min \text{IoU}_{(2)} = \begin{cases} \frac{(x_1 - \varepsilon_{\text{IoU}})(y_1 - \varepsilon_{\text{IoU}})}{xy} & \text{if } x_1 < x, y_1 < y \\ \frac{xy}{(x_1 + \varepsilon_{\text{IoU}})(y_1 + \varepsilon_{\text{IoU}})} & \text{if } x_1 > x, y_1 > y \\ \frac{1}{\frac{x_1 + \varepsilon_{\text{IoU}}}{x} + \frac{y}{y_1 - \varepsilon_{\text{IoU}}} - 1} & \text{if } x_1 > x, y_1 < y \\ \frac{1}{\frac{y_1 + \varepsilon_{\text{IoU}}}{y} + \frac{x}{x_1 - \varepsilon_{\text{IoU}}} - 1} & \text{if } x_1 < x, y_1 > y \end{cases}$$

The demonstration was performed for $x > 0, y > 0$. It is also applied to four quadrant by symmetry if we consider the origin in the center of $B_2 \cap B$.