Qualification Considerations of Machine Learning Based Tools for Avionics System Development

Cong Liu, Heber Herencia-Zapana, Scott Nagel, Kyle Ford, Darren Cofer
Collins Aerospace
{first.last}@collins.com

***Abstract*—Machine learning (ML) technology has advanced significantly in recent years, enabling its practical application in various domains, including avionics. However, before ML can be integrated into avionics systems, it must comply with certification or qualification standards. While much attention has been given to the certification of ML applications, the qualification of ML-based tools has been less extensively studied. This paper explores the qualification of ML-based tools in avionics system development, examining the unique characteristics of ML and their impact on tool qualification. We propose a methodology for qualifying low-criticality ML-based tools, aligned with the DO-330 software tool qualification standard, treating the ML tool as a black box. To demonstrate this approach, we present a case study of a ML-based avionics display testing tool.**

***Index Terms*—machine learning, tool qualification**

I. Introduction

The recent rapid advancements of artificial intelligence (AI) and machine learning (ML) have attracted a lot of interests in using this technology in the aerospace domain. This includes the next-generation airborne collision avoidance system for unmanned aircraft (ACAS Xu) [1], airport runway detection for autonomous vision-based taxi, takeoff, and landing systems [2] and vertiport detection for precision navigation in urban air mobility (UAM) [3]. However, airborne systems with AI/ML component must demostrate compliance with rigorous certification standards. The novelty of the ML technology brings great challenges, as certain aspects of the existing standards could be either inapplicable or insufficient.

To address the certification concerns, the Federal Aviation Administration (FAA) has published a techinical concept paper [4] and the European Union Aviation Safety Agency (EASA) has published several guidances [5] [6]. However, these documents so far have been focused on ML applications. There is a lack of clear guidance on the qualification of ML-based tools.

Tool qualification is the process by which certification credit maybe claimed for the use of a software tool. ML-based tool qualification presents unique challenges and opportunities in safety-critical systems development. Unlike traditional software, ML algorithms are often developed using data-driven processes, making their behavior difficult to understand. Qualification of ML-based tools may require a tailored approach that addresses data integrity, model training, and performance consistency under varying conditions. Regulatory frameworks, such as DO-330, must be carefully interpreted and extended to accommodate the characteristics of ML. As the adoption of ML continues to grow, a rigorous and evolving qualification methodology is essential to ensure compliance, safety, and trust in these advanced technologies.

II. Background

A. Machine Learning Tools

The RTCA DO-178C standard [7] defines a software tool as a computer program used to assist in the development, testing, analyzing, producing, or modification of another program. In the development of avionics systems, tools play a critical role at every stage, from initial design to final validation. ML is a field of AI where neural network models are trained to identify patterns or make decisions without being explicitly programmed for every task. ML represents a paradigm shift in software development, moving from explicitly rule-based instructions to systems that learn behaviors from data. We classify the tools related to ML into two categories.

- The first category contains the tools used in the ML application development lifecycle (MLDL) [16]. These tools themselves are often traditional software. This includes data collection tools, synthetic data generation tools, data processing tools, ML training tools, ML optimization tools and ML verification tools.
- The second category contains the tools that have a ML component. These tools are often used in a traditional software or system development lifecycle. This includes generative-AI tools (e.g., used to generate source code, formal specification or test cases), ML-based computer vision tools and ML-based speech recognition tools.

We refer the second category of tools as *ML-based tools*. This paper mainly discusses the qualification of ML-based tools.

B. Tool Qualification Standards

RTCA DO-330 "Software Tool Qualification Considerations" [8] defines the objectives and processes required to qualify a software tool used in the development of safety-critical systems. It has been widely used as the tool qualification standard for airborne systems. We will discuss it in more details in Section III.

ISO/DIS 26262 "Road vehicles — Functional safety" [9], is an international standard that establishes guidelines for the safe design and development of electrical and electronic systems in road vehicles. Part 8 "Supporting Processes" of the standard provides detailed guidance on tool qualification. The

tool qualification process starts with Tool Confidence Level (TCL) determination. First, the tool impact is analyzed. Only if a tool could insert or fail to detect an error, tool qualification applies. Then the probability of the tool error being detected in the subsequent process is analyzed. Higher probability of error detection means higher degree of confidence (i.e., lower TCL). The required TCL, together with the Automotive Safety Integrity Level (ASIL) of the software developed or verified using the tool, determines the appropriate tool qualification methods. Compared to DO-330, ISO 26262 does not distinguish between development or verification tools. Recent advances in ML for automotive applications have led to the development of techniques [10] to enable ML-based systems to comply with ISO 26262.

RTCA DO-200 "Standards for Processing Aeronautical Data" [11] provides guidance and requirements for the processing of aeronautical data that are used for navigation, flight planning, terrain/obstacle awareness, flight deck displays, flight simulators and for other applications. It is an adpation [12] of DO-330 to aeronautical databases. DO-200 has been proposed [13] [14] as a basis for ML data management tool qualification.

III. DO-330 Tool Qualification

DO-330 tool qualification begins with establishing the need for qualification. Qualification is required if:

- the tool is used to eliminate, reduce, or automate a software life cycle process, and
- the tool's output is not subsequently verified.

DO-330 introduces the concept of Tool Qualification Level (TQL), with each level associated with a specific set of qualification objectives that the tool must satisfy. The determination of the TQL depends on both the criticality of the software the tool supports and the extent of the tool's impact on the software development process.

Tool Impact Criteria Based on tool impact, DO-330 classifies tools into three categories, as shown in Figure 1. The first criteria applies to tools that could introduce errors into the resulting software and are usually referred to as *development tools*. Examples include source code generators and compilers. The second and third criteria apply to tools that could fail to detect errors in the software and are often referred to as *verification tools*. Examples include test generation tools and static code analysis tools. There are subtle distinctions between Criteria 2 and Criteria 3. Criteria 2 applies when a verification tool is used to eliminate or reduce a verification or development process other than the one automated by the tool itself. For example, a formal methods tool used to automate source code verification would meet Criteria 2. Due to the higher qualification rigor and cost associated with Criteria 2, applicants typically avoid such claims, and most verification tools are qualified under Criteria 3.

Tool Qualification Level Determination The software level [7] is established through the safety assessment process and hazard analysis by evaluating the potential impact of the failure conditions. There are 5 levels: catastrophic (A), hazardous (B), major (C), minor (D), no safety effect (E). Based on the airborne software level and the tool qualification criteria, DO-330 defines 5 tool qualification levels (TQL), as shown in Figure 2. They are denoted as TQL-1, TQL-2, TQL-3, TQL-4 and TQL-5, from the most to the least rigorous. Note that Criteria 3 tools are always qualified at TQL-5 no matter what assurance level of the airborne software is. In practice, most tools are qualified as TQL-5 to minimize qualification effort and cost [15]. Our discussion is focused on qualification of ML-based TQL-5 tools.

Criteria 1: A tool whose output is part of the resulting software and thus could insert an error.
Criteria 2: A tool that automates verification process(es) and thus could fail to detect an error, and whose output is used to justify the elimination or reduction of: a. Verification process(es) other than that automated by the tool, or b. Development process(es) that could have an impact on the airborne software.
Criteria 3: A tool that, within the scope of its intended use, could fail to detect an error.

Fig. 1: DO-330 Tool Qualification Criteria. ©RTCA, Inc. Used with Permission. All rights reserved.

Software Level	Criteria		
	1	2	3
A	TQL-1	TQL-4	TQL-5
B	TQL-2	TQL-4	TQL-5
C	TQL-3	TQL-5	TQL-5
D	TQL-4	TQL-5	TQL-5

Fig. 2: DO-330 Tool Qualification Levels. ©RTCA, Inc. Used with Permission. All rights reserved.

DO-330 defines tool qualification objectives in 11 processes: tool operational process, tool planning processes, tool development processes, verification of outputs of tool requirements processes, verification of outputs of tool design processes, verification of outputs of tool coding and verification processes, testing of outputs of integration processes, verification of outputs of tool testing, tool configuration management processes, tool quality assurance processes and tool qualification liaison processes.

For TQL-5, there are in total 14 tool qualification objectives. They include:

- Establish tool qualification need
- Define Tool Operational Requirements (TORs)
- Demonstrate tool operation compliance with TORs
- Demonstrate TORs sufficiency and correctness
- Identify and analyze the impact of known problems on TORs

A. Tool Operational Requirements

TORs define the tool's functionality and interface from a software life cycle process perspective. TORs must be

verifiable, consistent, and detailed enough to show the tool is functionally equivalent to the process it is replacing. DO-330 suggests TQL-5 TORs to include:

- Description of the context of the tool and its integration
- Description of Tool Operational Environment
- Description of tool input and output
- Requirements for the functionality of the tool
- The relevant user information, e.g., a user manual
- Description of the operation of the tool
- Performance requirements specifying the output behavior

IV. Qualification Considerations for ML-based Tools

Tool Qualification Needs If the output of a ML tool is manually reviewed or verified, the tool does not require qualification. For example, a large language model (LLM)-based code generation tool used in software development would not need to be qualified if its generated code is subject to manual review. Similarly, an ML training tool does not require qualification, as its output, the trained ML model, is verified against MLDL requirements. In general, validating the outputs of ML tools offers a cost-effective alternative to full-scale tool qualification. Adopting this strategy can substantially reduce the ML tool qualifcation needs in practice.

Tool Impact Criteria It has been proposed [14] to modify the tool impact criteria in order to better accommodate ML tools. While the underlying motivation for this change is understandable, such modifications may introduce unnecessary complexity to the qualification process of ML tools. ML tools remain compatible with the three established tool categories. Therefore, we propose maintaining the existing DO-178C/DO-330 tool impact criteria without any modifications.

Dataset Supervised learning is a widely used ML technique that employs labeled datasets to train neural network (NN) models for outcome prediction and pattern recognition. Data plays a critical role in both the development and verification of ML models. However, there is no clear guidance on how datasets should be treated within the ML development lifecycle. Key questions remain, such as whether training data should be regarded as requirements, how to ensure data quality (e.g., absence of data poisoning), and how to guarantee that datasets are complete and representative. Additionally, many foundational NN models are pre-trained using diverse data sources and human-in-the-loop training methodologies, making it extremely difficult, if not impossible, to fully identify all the datasets used during training.

Traceability Traceability is a fundamental concept in DO-330. Specifically, the "Tool Development Process Traceability" section requires trace data demonstrating a bi-directional association between low-level tool requirements and the tool source code. For ML-based tools, the training dataset is often treated as the "low-level requirements," while the resulting ML model (e.g., a neural network) is considered the source code. However, due to the inherent nature of machine learning, it is highly challenging to establish traceability between specific components of the model—such as individual layers or neurons—and the particular data samples that influenced their parameters. Furthermore, a recent trend in ML development involves a two-phase process, where models (often large language models) are first pre-trained on large, general datasets and later adapted using domain-specific datasets. In such cases, detailed information about the pre-training datasets is typically unavailable, resulting in a complete loss of traceability from the perspective of the ML-based tool end user. In conclusion, ML-based tools inherently lack the source code traceability expected in traditional software development.

Determinism In an early version of DO-178, the guidance explicitly stated that 'only deterministic tools may be qualified.' However, the term 'deterministic' was later deemed overly restrictive. As a result, the language was revised to adopt a more flexible and inclusive phrasing, allowing for a broader interpretation of tool behavior within the qualification process. Most ML models operate deterministically, producing consistent outputs for identical inputs. However, certain generative AI models, such as LLMs, incorporate elements of randomness to enable variability in responses to the same input. This controlled randomness is designed to emulate aspects of human creativity and spontaneity. It is typically governed by configurable parameters, such as the 'temperature' setting, which influence the degree of output variation. When the temperature is set to a low value (e.g., zero), the model behaves deterministically, generating the most probable response consistently. Conversely, higher temperature values introduce greater variability, supporting more diverse and creative outputs. For tool qualification purposes, we propose to require the ML model to be completely deterministic.

Explainability The internal operation of ML models is difficult to understand, and their behavior is normally characterized probabilistically. This makes it difficult for humans to reliably predict or fully explain their outputs. This lack of explainability necessitates treating ML-based tools as "black boxes" during qualification and verification. While generative AI models, such as LLMs, can produce explanations as part of their outputs, these explanations are often unreliable and cannot be fully trusted to reflect the true reasoning behind the model's decisions.

Generalization Error No ML model can guarantee 100% accuracy on previously unseen inputs. The probability of generalization error always exists. This makes it essential to detect and mitigate tool errors. However, DO-330 does not provide guidance on tool error detection or mitigation. Extending DO-330 to address this aspect would significantly enhance its applicability to the qualification of ML-based tools.

V. Use Case

A ML-based tool was developed to automate the testing of an avionics display software. Traditionally, a test engineer is needed to monitor the display to verify that certain features are displayed correctly (e.g., a specific button is selected). The manual verification is tedious, time-consuming and expensive. The tool was developed to replace the human. The core of the

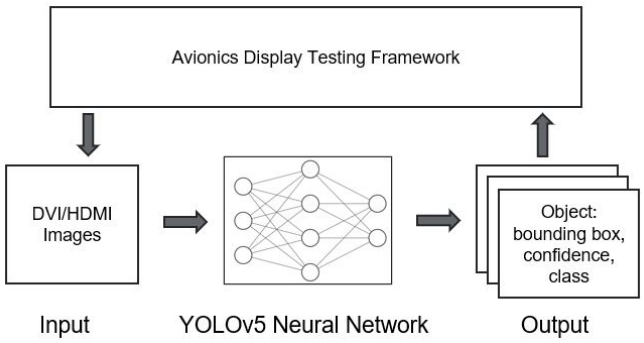

Fig. 3: Overview of the ML-based Display Testing Tool

ID	Requirement
TOR-1	The tool shall only be used to test the intended Avionics display.
TOR-2	The tool shall use less than 16GB memory.
TOR-3	The tool shall operate without a GPU.
TOR-4	The tool shall accept input image formats: JPEG, PNG, BMP, TIFF, GIF.
TOR-5	The tool shall accept input images in either RGB or Grayscale color mode.
TOR-6	The tool shall accept the original image without resizing or padding.
TOR-7	The tool shall accept user specified confidence threshold (a number between 0 and 1).
TOR-8	The tool shall report "pass" if the computed confidence score of the feature (class) to be verified is greater than the specified confidence threshold. Otherwise, it shall report "fail".
TOR-9	The tool shall generate output within 10 seconds.
TOR-10	The tool shall produce the same output given the same input.

Fig. 4: Tool Operational Requirements

tool is an object detection and classification neural network. As shown in Figure 3, the input to the NN is a undistored image captured at the DVI/HDMI interface. The output of the NN is the classification, indicating what features are present in the image. The NN, based on a pre-trained open-source YOLO (You Only Look Once) model [16], was fine-tuned using a domain specific image dataset.

Tool Qualification Need This tool replaces an existing verification process and its output will not be subject to manual review or verification. Therefore, qualification of the tool is required.

Tool Qualification Level Since the tool automates the testing of a display software, it has been categorized as a verification tool. Furthermore, it does not eliminate or reduce any other development or verification activities beyond those specifically automated by the tool itself. Based on this analysis, Tool Impact Criteria 3 was determined to be applicable. Domain experts assessed the software level of the display software as Level D. Consequently, the tool was assigned TQL-5.

Tool Qualification Objectives It is important to recall that TQL-5 tools have no qualification objectives related to tool development processes. As a result, whether the tool was created using a data-driven approach or a traditional software development method is irrelevant for qualification purposes. Since the objectives are focused solely on tool usage, the tool may be treated as a "black box" by the applicant. Recognizing this implication, we beleive that the qualification of TQL-5 ML-based tools is fundamentally equivalent to that of traditional software tools.

A. Tool Operational Requirements

Defining Tool Operational Requirements (TOR) is key to TQL-5 tool qualification. But it presents several challenges:

- Distinction between TOR and Tool Requirements (TR): It is necessary to clearly differentiate TOR from TR and determine which requirements should be included in the TOR, the TR, or both.
- Interpretation of Performance Requirements: A thorough understanding of the "performance requirements" referenced in the standard is required to ensure appropriate specification within the TOR.
- Level of Detail in TOR: TOR can be defined at varying levels of granularity. If written at too high a level, it complicates verification activities; if too detailed, it may include irrelevant information not essential to tool operation.
- Identification of Concrete Requirements: Given that the term "requirements" explicitly appears only in some TOR items in the standard, while the others serve as descriptive information. It is necessary to determine, for ML-based tools, which elements should be explicitly included in the TOR.
- Completeness of the TOR Definition: It is difficult to ascertain whether the TOR has been fully and sufficiently defined.
- Extension of TOR for ML Tools: ML-based tools may require additional TOR elements beyond those listed in the standard.

Some sample TORs are shown in Figure 4.

The TORs are defined based on the following considerations:

- Intended Scope of Use: ML models are typically trained and validated for specific applications. Therefore, the ML-based tool must be restricted to operation within its defined scope of use.
- Operational Environment Requirements: Certain ML models require advanced hardware resources (e.g., GPUs, specialized accelerators) to achieve acceptable performance. Such environmental requirements can be critical to correct tool operation, and thus must be explicitly specified in the TOR.
- Input Data Constraints: Object detection ML models often assume specific input image formats and color modes. These assumptions must be documented to ensure proper tool operation .
- Impact of Confidence Level: The confidence threshold is a critical parameter that directly affects the tool's output. The expected behavior of the tool may vary depending on the configured confidence level, and this dependency must be captured in the TOR.
- Timing Requirements: When an ML tool is integrated as a component or step within a larger process, it is important

to specify the expected timing of its output. Given the potential performance variability of ML models, execution time requirements must be explicitly included in the TOR.
- Deterministic Behavior: Finally, the ML model is required to exhibit deterministic behavior. Non-deterministic tools are difficult to verify and complicate the reproduction of results, making determinism essential for qualification.

Tool Usage Context The tool shall be integrated with the avionics display software test framework. A statement in the test script shall pause the test execution and trigger the execution of the tool. At the end of the execution, the tool shall record the verification results and return control to the test framework to allow the test to resume.

Tool Operational Environment The tool shall be installed on a Linux or Windows operation system with Python version 3.8.0 or later and PyTorch version 1.8 or later. All required Python packages shall be installed.

User Information A user manual shall be provided to describe the intended usage within the test framework. It shall describe the input and output of the tool. An installation guide shall be provided to describe step-by-step instructions on installing Python, PyTorch and all required Python packages.

Performance Requirements DO-330 does not provide an explicit definition of performance requirements. According to ARP4754A [17], performance requirements specify the attributes of a function or system that contribute to its utility for the aircraft and its operation. In addition to describing the expected type of performance, performance requirements encompass function-specific characteristics such as accuracy, fidelity, range, resolution, speed, and response time. An example of a performance requirement is illustrated by TOR-9 in Figure 4.

B. Operational Design Domain

According to the EASA guidance [5], an ML application must have a clearly defined Operational Design Domain (ODD). Specifically, Objective DM-01 states: "The applicant should define the set of parameters pertaining to the AI/ML constituent ODD." The ODD delineates the specific conditions and environments under which the ML tool is designed to operate reliably and as intended. In the context of ML tool qualification, establishing the ODD is essential to impose constraints and requirements on the dataset used for verification activities, ensuring that the tool is evaluated within its intended operational scope. For the ML tool under consideration, the ODD is defined as the set of display images that the tool is expected to inspect.

We conducted a detailed analysis of the display image format and structure. The display is organized into three primary regions: the top, middle, and bottom areas. The main interactive content is located within the middle area, while the top and bottom areas contain static images that remain unchanged during testing. Within the middle region, the interface comprises four tabs, each containing multiple boxes, and each box, in turn, hosting a set of buttons. There are three distinct types of buttons, and both tabs and buttons can exist in two possible states: selected (indicated by a green outline) or unselected. Each combination of button/tab type and state was mapped to a specific class. For the buttons, six distinct classes were defined, corresponding to the two states (selected or unselected) across the three button types. Test cases were systematically generated by permutating the various combinations of button and tab states, while adhering to constraints derived from the graphical user interface (GUI) and display software requirements. For example, constraints such as "at most one mic button is selected at any given time" were enforced. A Simulink model was developed to formally capture these constraints and systematically generate all feasible system states. Subsequently, an automated image generation tool was employed to create display images of all valid state permutations.

C. ML-based Tool Limitation and Mitigation

ML-based tools have limitations. We highlight two issues below.

Generalization Error It is widely recognized that an ML model can produce incorrect outputs. A significant body of research, particularly in the area of adversarial attacks, is dedicated to identifying inputs that intentionally cause an ML model to fail or misbehave. This vulnerability highlights the broader issue known as the generalization problem, wherein the model's behavior cannot be reliably guaranteed outside the set of inputs used during training, validation, or testing. In practice, it is extremely challenging, if not impossible, to ensure that an ML model will consistently produce correct outputs for previously unseen inputs.

Overconfidence Many classification ML models generate a *confidence* value, ranging from 0 to 1. Higher values indicate that the model is more confident that the classification result is "correct." This value can also be interpreted as the model's estimated probability that the classification output is accurate. In the TORs, users are required to specify a confidence threshold for the tool. The tool will output a "pass" result only when the computed confidence exceeds the user-specified threshold. However, it is important to recognize that ML tools can sometimes exhibit overconfidence, meaning that even if the model reports a high confidence value, the classification output may still be incorrect. In other words, a high confidence score does not guarantee the accuracy of the classification result.

To address the tool limitations, we propose the following techniques to mitigate the error-induced risk.

Avoid Underfitting Underfitting occurs when a model is too simple to capture the underlying patterns within the data. This may result in elevated errors across both the training and test datasets. We propose to review the ML model architecture with domain experts and use a state-of-the-art (SOTA) ML model architecture that has demonstrated good performance in common benchmarks.

Avoid Overfitting Overfitting occurs when a model is excessively trained so that it is extremely accurate on the training dataset. This may prevent the model from capturing the actual patterns, thus reducing its ability to generalize to new, unseen data. We propose to use various techniques to avoid overfitting. These techniques include:

- Provide a complete and representative dataset
- Split data into training, validation and testing sets
- Use dropout layers during training
- Stop training once the model's performance plateaus on the validation set

Adversarial Training Adversarial training is a technique where the model parameters are iteratively updated to minimize the worst-case adversarial loss by incorporating adversarial examples into the training process. Adversarial training improves the model robustness against adversarial inputs.

Ensemble Learning Ensemble learning is a training technique in which two or more ML models are applied to a specific classification or regression task. This approach is based on the premise that combining multiple models can yield superior predictive performance compared to any single constituent learning algorithm. For tools with a high TQL, we recommend developing two distinct ML models, each utilizing different architectures, training algorithms, and datasets. This strategy helps mitigate the risk of common-source errors. The output will only be accepted if both models, when given the same input, produce consistent results.

VI. Generalization Bound

Generalization bounds are mathematical statements that help assess how well a ML model can generalize. In other words, they measure how accurately the ML model is expected to predict the output for an input it has never encountered before. To define this measure, let's first introduce some basic concepts. Let x represent an input and $F(x)$ denote the corresponding output, where F defines the relationship between inputs and outputs and it is often unknown. The primary goal of ML training is to develop a model h that approximates this relationship F. Due to the probabilistic nature of model training, there is always some error associated with a model h for a given input x. This means that for an input x, $h(x)$ will be close to, but not exactly equal to, $F(x)$. This error can be expressed as a mathematical function m, which compares the predicted value $h(x)$ with the actual output $F(x)$, measuring how close they are. This is represented as $m(h(x), F(x))$. To evaluate how well the model h performs, we need to measure the error across all the inputs in the operational domain where the model is expected to operate. This is known as the Operational Domain Design (ODD). The overall error of the model h across the ODD is represented as $E(m(h(x), F(x)))$, which refers to the expected error of the model considering all inputs in the ODD. This expected error is called the out-of-sample error, e_{out}. A key characteristic of e_{out} is that calculating it requires knowing all the inputs and outputs $(x, F(x))$ within the ODD, which can be impractical or impossible. However, it is possible to compute an approximation using a subset of N inputs and outputs, and calculate the expression $\frac{1}{N}\sum_{i=1}^{N} m(h(x_i), F(x_i))$, which is called the in-sample error, e_{in}. The question now is whether, given the value of the in-sample error, it's possible to estimate the out-of-sample error. This question is addressed by the concept of the generalization gap. It states that with probability greater than $1-\delta$, the out-of-sample-error is ϵ-close to the in-sample-error, as shown in the equation below [5], [18], [19].

Theorem 1 (Generalization Bound): If there are N identically and independently distributed random variables $m(h(x_i), F(x_i))$, which are bounded on the closed interval $[a, b]$, then the following inequality holds:

$$\Pr\left(|e_{in} - e_{out}| \leq \epsilon\right) \geq 1 - \delta,$$

where $\delta = 2e^{\frac{-2N\epsilon^2}{(b-a)^2}}$, $e_{out} = E(m(h(x), F(x)))$, $e_{in} = \frac{1}{N}\sum_{i=1}^{N} m(h(x_i), F(x_i))$.
There are several versions of this generalization bound theorem, and the paper [20] provides a comprehensive list of the existing versions, with Theorem 1 being the simplest and most general version found in the literature. For instance, it does not account for the complexity of the ML model architecture, but it is still useful for the types of problems addressed in this paper. One way to interpret this theorem is that, with a probability greater than $1-\delta$, the out-of-sample error falls within the interval $[e_{in} - \epsilon, e_{in} + \epsilon]$. An important takeaway from this expression is how the generalization gap behaves. For example, if a high level of accuracy is desired, meaning the interval should be narrower (i.e., making ϵ very small), the lower bound $1-\delta$ decreases toward zero. This implies that there is less confidence that the out-of-sample error will lie within the interval. One way to boost confidence is by increasing the number of samples, N, though this can be costly or impractical. To achieve the right balance between acceptable confidence, accuracy, and a reasonable sample size N is essential when using the generalization gap for safety analysis. In some cases, finding this balance can be tricky, but the key insight is that it depends on the specific problem at hand. The next subsection will illustrate how this theorem applies to a particular problem.

VII. Generalization Bound Application

This section demonstrates the application of the generalization gap theorem to analyse the safety impact of the ML errors of the use case tool. The primary motivation is to evaluate the safety implications of incorporating ML to show the complaince of DO-178C objectives [7]. The safety impact is assessed using a confusion matrix, which highlights two types of errors—false negatives and false positives—that could compromise the Means of Compliance (MOC) of Objective 2 in Table A-7 of DO-178C, as shown in Table I.

False negatives do not impact the MOC because if the test passes and the YOLO-based tool erroneously classifies it as a failure, DO-178C, Table A-7, Objective 2, states that "the objective is to ensure that the test results are correct

	Test:Pass	Test:Fail
YOLO:Pass	True positive	False positive
YOLO:Fail	False negative	True negative

TABLE I: Confusion Matrix

and that discrepancies between actual and expected results are explained." This means that the test results must be manually reviewed. During the review, it will be identified that the error originated from the YOLO-based tool. While this increases the workload of the testing process, it does not compromise the MOC. On the other hand, false positives compromise the MOC because if the YOLO-based tool declares that the test passes when it actually fails, the tool does not provide evidence to support this conclusion, and such evidence is not required by DO-178C. This could create a misleading impression that the GUI passed the test and met the DO-178C objective 2 from Table A-7, when in reality, it did not. As a result, there is a risk of falsely believing that the GUI achieved the required Design Assurance Level (DAL) when, in fact, it did not. The key question is: Given that False Positive error is inevitable, what is the acceptable likelihood of this error to guarantee the desired DAL for the GUI? To address this question, we will break it down into two parts: First, Error Calculation, where we define and calculate the false positive error of the YOLO-based tool that could compromise the GUI's desired DAL. Second, Error and Safety Analysis, where we identify the conditions under which the calculated YOLO-based tool false positive error does not undermine the GUI's desired DAL.

A. Error Calculation

The safety concern arise from the potential for False Positive to undermine the MOC of Objective 2 from Table A-7. Mathematically, False Positive can be represented by an indicator function, as shown Equation (1), which takes the value of one if the test fails ($F(x)$: Fail) while the YOLO-based tool incorrectly declares a pass ($h(x)$: Pass), and zero otherwise.

$$m(h(x), F(x)) = \begin{cases} 1 & \text{if } h(x)\text{: Pass and } F(x)\text{: Fail} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

With this mathematical expression of $m(h(x), F(x))$, we can apply Theorem 1 to this context, where the out-of-sample error corresponds to the probability of False Positive, and the in-sample error is the ratio of false positives to the total number of samples. This leads to the instantiation of Theorem 1 as follows.

Theorem 2 (Generalization Bound Instantiation): If there are N identically and independently distributed random variables $m(h(x_i), F(x_i))$ then the following inequality holds:

$$\Pr\left(|\mathrm{P} - \Pr(\text{False positive})| \leq \epsilon\right) \geq \delta',$$

where $\mathrm{P} = \frac{\text{Number of false positives}}{N}$ and $\delta' = 2e^{-2N\epsilon^2}$.
One way to interpret Theorem 2 is that, with a probability greater than $1-\delta'$, the out-of-sample error, $\Pr(\text{False positive})$, falls within the interval $[\mathrm{P} - \epsilon, \mathrm{P} + \epsilon]$.

B. Error and safety analysis

This subsection outlines conditions that should be met regarding the YOLO-based tool false positive error to ensure that the desired GUI DAL is not compromised. The YOLO-based tool's false positive error can negatively impact the MOC of Objective 2 from Table A-7 in DO-178C, potentially leading to a situation where the YOLO-based tool incorrectly assesses that Objective 2 has been achieved. This could create the illusion that the required likelihood of failure for the GUI has been met, while in reality, the likelihood of a GUI error is higher because it was concealed by the false positive. Essentially, the error hides a GUI failure, falsely labeling it as a pass when it actually did not meet the required criteria. Now, the key issue is how to ensure that the likelihood of the tool false positive does not conceal the true likelihood of a GUI error. One way to guarantee this is by ensuring that the YOLO-based tool's false positive error is less than the required likelihood of the GUI DAL. This can be achieved using Table II and Theorem 2 as follows: Table II provides the required likelihood for a specific DAL, while Theorem 2 gives the upper bound for the unknown probability of false positives with a probability bigger than $1 - \delta$, which depends on the in-sample error P and the accuracy ϵ as follows $\Pr(\text{False positive}) < \mathrm{P} + \epsilon$. So, together Table II and this upper bound lead to the values in Table III. These inequalities not only ensure that if a failure occurs, it is due to the GUI's required failure likelihood and not the YOLO-based tool's false positive error, but also guarantee that the YOLO-based tool's false positive error does not compromise the desired GUI DAL.

DAL	**SEVERITY**	**LIKELIHOOD**
A	Catastrophic	Extremily improbable $]0, 10^{-9}]$
B	Hazardous	Extremely remote $[10^{-9}, 10^{-7}]$
C	Major	Remote $[10^{-7}, 10^{-5}]$
D	Minor	Probable $[10^{-5}, 10^{-3}]$

TABLE II: Relationship between probability and severity of failure conditions (FAA AC-25-1309) [21]

DAL	**In-sample False positive error**
A	$P + \epsilon < 10^{-9}$
B	$P + \epsilon < 10^{-7}$
C	$P + \epsilon < 10^{-5}$
D	$P + \epsilon < 10^{-3}$

TABLE III: In-sample error inequalities

In summary, if the YOLO-based tool satisfies the criteria for Criterion 3, and statistical analysis demonstrates that Table III has been met for the required GUI DAL, then this approach provides sufficient evidence to support the application for tool qualification. This is because Criterion 3 treats the ML tool as a black box, meaning evidence related to the ML tool's training is not required. The primary task is to show that the system's DAL is not compromised by the ML tool, which can be justified using Table III.

VIII. Related Work

Although there have been a lot of discussion about the certification of ML applications, the qualification of ML-based tools got little attention from the community until recently.

The qualification of tools used in ML application developement was discussed in [13]. The approach is mainly follows DO-330 and tailored the tool classification and leveling to align with DO-200. It proposed to extend the DO-178C impact criteria for ML tools. Although we understand the motivation of aligning ML tool qualification with DO-200, we argue that the data used in ML developement is much more critical than the aeronautical databases. The data directly impacts the ML application. Thus, the tool could insert an error to the airborne software.

The tool qualification of generative AI and synthetic data generation was discussed in [14]. A new tool qualification criteria was proposed for generative AI tools. The DO-200 is used as a reference to justify the proposal. We think it is important to understand the difference between aeronautical databases and airborne system databases [12]. They are similar. But due to different safety impact, they are approved under DO-200 and DO-178, respectively. If the generative AI and synthetic data generation tools could insert an error to the airborne software, it is reasonable that they should be subject to the same stringent requirements as the traditional development tools.

IX. Conclusion

Low-criticality (e.g., TQL-5) ML-based tools can be qualified with the existing DO-330 framework. This is because DO-330 TQL-5 qualification imposes no requirements on tool development process. The qualification is focused on tool usage and operation. Characteristics of ML (e.g., non-determinism, generalization error) must be carefully considered and counted for in the qualification.

References

[1] K. D. Julian, J. Lopez, J. S. Brush, M. P. Owen, and M. J. Kochenderfer, "Policy compression for aircraft collision avoidance systems," in *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, 2016, pp. 1–10.

[2] R. A. Amit and C. K. Mohan, "A robust airport runway detection network based on R-CNN using remote sensing images," *IEEE Aerospace and Electronic Systems Magazine*, vol. 36, no. 11, pp. 4–20, 2021.

[3] E. Miccio, P. Veneruso, R. Opromolla, G. Fasano, C. Tiana, and G. Gentile, "Vision-aided sensing pipeline with AI-based vertiport detection for precision navigation in UAM approach and landing scenarios," Available at https://ssrn.com/abstract=5059951, 2024.

[4] T. Pham, "Verification of an artificial neural net model developed through machine learning (draft)," November 2024.

[5] EASA, "EASA Concept Paper: Guidance for Level 1&2 machine learning applicationsn," https://www.easa.europa.eu/en/document-library/general-publications/easa-artificial-intelligence-concept-paper-issue-2, March 2024.

[6] ——, "EASA Concept Paper: First usable guidance for Level 1 machine learning applications," April 2021.

[7] RTCA DO-178C, "Software Considerations in Airborne Systems and Equipment Certification," December 2011.

[8] RTCA DO-330, "Software Tool Qualification Considerations," December 2011.

[9] ISO 26262, "Road vehicles — Functional safety," 2018.

[10] J. Serna, S. Diemert, L. Millet, R. Debouk, R. S, and J. Joyce, "Bridging the gap between ISO 26262 and machine learning: A survey of techniques for developing confidence in machine learning systems," *SAE International Journal of Advances and Current Practices in Mobility*, vol. 2, no. 3, pp. 1538–1550, 2020.

[11] RTCA DO-200C, "Standards for Processing Aeronautical Data," June 2024.

[12] J. Marques and A. M. da Cunha, "Use of RTCA DO-330 in aeronautical databases," in *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*, 2015, pp. 1–18.

[13] K. Dmitriev, F. Kaakai, M. Ibrahim, U. Durak, B. Potter, and F. Holzapfel, "Tool qualification aspects in ML-based airborne systems development," in *Software Engineering 2023 Workshops*. Bonn: Gesellschaft für Informatik e.V., 2023, pp. 208–221.

[14] H. Glenn Carter, C. Vinegar, V. Terres, and J. Rupert, "Considerations for tool qualification in flight-critical applications using machine learning," in *2024 AIAA DATC/IEEE 43rd Digital Avionics Systems Conference (DASC)*, 2024, pp. 1–10.

[15] M. Ibrahim and U. Durak, "State of the art in software tool qualification with DO-330: A survey." *Software Engineering (Satellite Events)*, pp. 1–23, 2021.

[16] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[17] SAE ARP4754A, "Guidelines for Development of Civil Aircraft and Systems," December 2010.

[18] EASA, "Concepts of design assurance for neural networks (CoDANN)," 2020. [Online]. Available: https://www.easa.europa.eu/en/document-library/general-publications/concepts-design-assurance-neural-networks-codann

[19] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning from data*. AMLBook New York, 2012, vol. 4.

[20] M. Consortium, "EASA research – machine learning application approval (MLEAP) interim technical report," European Union Aviation Safety Agency, Horizon Europe research and innovation programme report, 5 2023.

[21] FAA, *AC 25-1309: Aircraft Systems and Equipment Certification*, Washington, D.C., 2004.