Final Report
AFE 87 – Machine Learning

Author(s): AFE 87 Project Members
Date: 7 May 2020
Issue: 1.0

Document ID: 87-REP-01
Release Status: Public
Approved: 5 June 2020

Aerospace Vehicle Systems Institute
754 HRBB - MS 3141
College Station, TX 77843-3141
Office: +1-979-862-2316
Web: www.avsi.aero

Document Revisions

REV	DATE	Author(s)	Modifications	Approved
1.0	5 June 2020	D. Redman / D. Ward / M. Carrico	Release version	5 June 2020

Acknowledgements

Matthew Carrico from Collins Aerospace led this project as the chair of the AFE 87 Project Management Committee (PMC). Dr. Donald Ward from AVSI provided project management support. Members of the AVSI AFE 87 Project that contributed to this research included:

Airbus	Bryce Durham, Jean-Claude Laperche
Boeing	Joshua Fadaie, Larry Olson, Steve Younger
Collins Aerospace	Eric Anderson, Matt Carrico, Pierre Dubois, Michael Giering, Tiffany Williamson, Radek Zakrzewski
Embraer	Allan Anderson, Ivo Medeiros, Paula Olivio, Fabricio Jose Pontes
FAA	Liz Brandli, Barbara Lingberg, Srini Mandalapu, Manny Rios, George Romanski, John Strasburger, Mike Vukas,
GE Aviation	Michael Durling, Ryan Gorby, Craig McMillan, Abha Moitra,
Honeywell	Pavan Allalaghatta, Devesh Bhatt, Ricardo Bortolini, Sai Krishnan Chandrasekar, Kevin Driscoll, Brendan Hall, Rakesh Jha, Anitha Murugesan, Hao Ren
NASA	Alwyn E. Goodloe, Kurt Woodham
Saab	Mats Ekman, Ingemar Söderquist, Månefjord Torbjörn, John Törnblom
Thales	Philippe Bougard, Christian Cantaloube, Marc Gatti
AVSI	Dr. David Redman, Dr. Donald Ward, Michael Kerstetter

Table of Contents

Document Revisions i
Acknowledgements ii
Table of Contents iii
List of Figures vi
List of Tables vii
List of Acronyms viii
Executive Summary x
1 Introduction 1
1.1 AFE 87 Project Description and Scope 2
1.2 Document Overview 2
2 Machine Learning Current State 3
2.1 Machine Learning Overview and Perspective 4
2.1.1 Unsupervised Learning 4
2.1.2 Supervised Learning 5
2.1.3 Reinforcement Learning 5
2.2 Deep Learning Overview 6
2.2.1 Artificial Neural Networks (High Level Perspective) 7
2.2.2 Artificial Neural Networks (Detailed Perspective) 8
2.3 Machine Learning Systems Development Cycle 9
2.4 Machine Learning Related Complexity Analysis 12
2.5 Machine Learning Technical and Industry Trends 13
2.6 Certification Challenges When Using ML in Safety-Critical Applications 14
3 Safety Considerations When Using Machine Learning 17
3.1 Datasets 19
3.2 Adversarial Inputs and Robustness 21
3.2.1 Definition of Robustness 23
3.2.2 Training Considerations for Improving Robustness 24
3.2.3 Testing of Robustness in Deep Learning Systems 26
3.2.4 Summary Observations and Research Needs 27
3.3 Observability and Interpretability 28
3.3.1 Relevance of Interpretability Techniques 28
3.3.2 Use Cases for Aviation 29
3.4 Research Summary - Safety Systems Using Machine Learning 30
4 Machine Learning Process Flow for Safety-Critical Applications 31
4.1 Introduction 31

4.1.1 ML Development Life Cycle Process 31
4.1.2 Requirements 32
4.2 Process Flow – Machine Learning as a Design Aid 33
4.3 Process Flow – Fixed Runtime Parameters 33
4.4 Process Flow – In Service Updates 34
4.5 Machine Learning Datasets Considerations 35
4.6 Design 36
4.7 Deployment 37
4.8 Validation Process 37
4.8.1 Validation Artifacts 38
4.8.2 Configuration Control Process 38
4.8.3 Quality Assurance Process 39
5 Perspectives on Assurance 40
5.1 Current Practice 40
5.1.1 Safety Assessment and Assumptions Related to Software Reliability 40
5.2 The Disruptive Nature of Machine Learning Implementations 41
5.3 Characterizing the Disruption 42
5.4 A Path Forward 43
6 Considerations for Bounding the Behavior of ML Algorithms 45
6.1 Runtime Assurance Architectures 45
6.1.1 Runtime Assurance Definition 45
6.1.2 Reference Architectures 46
6.1.2.1 Simplex vs. Complex System-Level Monitor 46
6.1.2.2 Tightly Coupled Runtime Monitor 46
6.2 Runtime Monitor Functions and Operations 47
6.3 Limitations of Using RTA Architectures 49
6.4 RTA Architecture Use Cases 50
7 Summary of Recommendations 52
7.1 Recommended Safety Objectives When Using ML in Safety-Critical Applications 52
7.1.1 Recommendations Related to Machine Learning Life Cycle: 52
7.1.2 Recommendations Related to Datasets: 52
7.1.3 Recommendations Related to Robustness 53
7.1.4 Recommendations Related to Safety Assurance 53
7.1.5 Recommendations Related to Runtime Assurance: 53
7.1.6 Recommendations Related to AI Interpretability: 54
7.2 Recommendations for Community Standards 54

7.3 Recommendations for Future AVSI Research 55
7.4 Conclusion 55
8 Glossary 57
Appendix A : Observability A-1
A.1 Dimensions of Interpretability A-1
A.2 Interpretability Definitions and Related Terms A-1
A.3 Logical Layers of Interpretability A-2
A.3.1 Technological Decomposition of Interpretability A-3
A.3.2 General Decomposition of Interpretability A-4
A.4 Interpretability Use Cases A-5
A.4.1 Visual Evidence Justifying Performance A-5
A.4.2 Self-Correcting Interpretable Systems A-6
A.4.3 Physics Guided Deep Learning A-7

List of Figures

Figure 1: Artificial intelligence, machine learning and deep learning. 3
Figure 2: Three primary branches of machine learning: Unsupervised, Supervised and Reinforcement Learning. 5
Figure 3: High level difference between machine learning and deep learning systems. 6
Figure 4: Schematic of an artificial neural network. 7
Figure 5: Common machine learning pipeline. 9
Figure 6: Automotive industry machine learning pipeline cycle. 11
Figure 7: Automotive industry's concept of operational design domain. 12
Figure 8: Automated software generation in computer vision. 14
Figure 9: Safety assurance guidance. 19
Figure 10: Example of an adversarial attack. 25
Figure 11: Literature search process. 30
Figure 12: Machine learning development flow. 32
Figure 13: Process flow: runtime fixed parameters. 34
Figure 14: Process flow: in service updates. 34
Figure 15: Dataset segregation for machine learning models. 35
Figure 16: Machine learning validation. 38
Figure 17: Configuration control. 39
Figure 18: Typical runtime assurance architecture. 45
Figure 19: System level monitor. 46
Figure 20: Reference model based assurance. 47
Figure A-1: Classifying the different approaches to exposing the black box. A-3
Figure A-2: Exposing a neural network by highlighting areas of an image. A-6
Figure A-3: Mechanism for defining attention-based heat maps via causal filtering. A-7
Figure A-4: A raw vibration signal decomposed into expected and "noise" components. A-8
Figure A-5: A deep auto-encoder network being trained. A-8

List of Tables

Table 1: Modeling aspects of machine and deep learning. 4
Table 2: ML training/testing & verification data criteria. 36
Table 3: Definitions of some runtime verification terminology. 48
Table A-1: Features utilized to decompose the technical approaches. A-4
Table A-2: General transparency levels to consider within an autonomous system. A-5

List of Acronyms

ACAS	Airborne Collision Avoidance System
ADS-B	Automatic Dependent Surveillance-Broadcast
AFE	Authorization for Expenditure (for AVSI Projects)
AI	Artificial Intelligence
AMC	Asynchronous Monitoring with Checkpoints
AMSD	Asynchronous Monitoring with Synchronous Detections
ANN	Artificial Neural Network
ARP	Aerospace Recommended Practice (SAE standard)
AVSI	Aerospace Vehicle Systems Institute
CA	Completely Asynchronous
CCA	Common Cause Analysis
CFR	Code of Federal Regulations
CMA	Common Mode Analysis
CS	Completely Synchronous
DARPA	U.S. Defense Advanced Research Projects Agency
DL	Deep Learning
DNN	Deep Neural Network
FAA	U.S. Federal Aviation Administration
FDAL	Function Development Assurance Level
FMEA	Failure Modes and Effects Analysis
FTA	Fault Tree Analysis
GDPR	General Data Protection Regulation
GPU	Graphics Processing Unit
HOL	Higher-Order Logic
IDAL	Item Development Assurance Level
IP	Intellectual Property
LIDAR	LIght Detection And Ranging
LRU	Line Replaceable Unit
LTL	Linear Temporal Logic
ML	Machine Learning
NN	Neural Network
ODD	Operational Design Domain
PMC	Project Management Committee
RAM	Random Access Memory
ReLU	Rectified Linear Units (activation function)
RESSAC	Re-Engineering and Streamlining Standards for Avionics Certification

RL	Reinforcement Learning
RTA	Runtime Assurance
RTCA	RTCA, Inc. (formerly Radio Technical Commission for Aeronautics)
RV	Runtime Verification
SAE	SAE International (formerly Society of Automotive Engineers)
SBML	System Based on Machine Learning
SGD	Stochastic Gradient Descent
SL	Supervised Learning
SMSI	Synchronous Monitoring with Synchronous Instrumentation
SSD	Solid State Drive
SVM	Support Vector Machine
TIAM	Technology Independent Assurance Method
TPU	Tensor Processing Unit
UL	Unsupervised Learning
V&V	Validation & Verification
XAI	eXplainable AI
ZSA	Zonal Safety Analysis

Executive Summary

This document summarizes the findings of the AVSI AFE 87 – *Machine Learning* project. The project was performed by AVSI members from Airbus, Boeing, Collins Aerospace, Embraer, FAA, GE Aviation, Honeywell, NASA, Saab, Thales, and AVSI. The project brought together subject matter experts with knowledge of aircraft systems certification and others with specific expertise in artificial intelligence (AI) and machine learning (ML) technologies. The objective was to investigate the safety and certification aspects of emerging machine learning technologies that are being incorporated in certifiable safety-critical aerospace systems. This report represents the primary output of the project, which provides AVSI members with useful information to begin introducing these technologies into future safety-critical platforms. Additionally, the recommendations in this report are intended to stimulate development of consensus standards, guidance, and assurance technologies by the broader set of aerospace industry stakeholders in order to rapidly adapt existing certification processes to accommodate these emerging technologies while maintaining the required level of safety.

The project was structured to address these issues through three major tasks: (1) a literature search, (2) consideration of a process flow specifically for machine learning systems, and (3) an investigation of suitable runtime assurance architectures. These tasks were specifically focused on highlighting certification/assurance issues and intentionally avoided detailed consideration of specific machine learning implementations. Furthermore, the subject matter expertise was largely focused on commercial transport applications, however the concepts are extensible to other applications as much as existing certification processes can be extended to these applications.

The initial task was a search of existing literature to understand the state of the art concerning safety assurance of machine learning technologies. The project found that while there is an extensive body of AI/ML literature, little research has been performed in the area of safety assurance. The investigation focused on artificial neural nets, as this provided the largest body of references, and this guided the focus of the other project tasks. This report includes extensive references and a full bibliography is available from AVSI upon request.

The second task was an investigation of the impact of ML technologies on existing certification processes and changes necessary to accommodate the fundamentally different nature of data-based systems. The project developed a reference ML development process flow (Section 4) that highlights the need to (1) properly manage data used to develop ML-based systems and (2) to incrementally validate and verify the system at each stage of the development life cycle. Importantly, the project concluded that initial application of ML in commercial aviation will be limited to systems that are trained during development and do not dynamically learn after being introduced into service.

The third task examined the applicability of runtime assurance architectures to bound the behavior of ML-based systems in a safety-critical system. The project recognized that such architectures may be useful to enable the introduction of these beneficial technologies into certified aircraft. The project concluded that simplex monitoring architectures could maintain adequate levels of safety provided that detection and interception of out-of-bounds behavior was possible within acceptable latency requirements.

Section 7 summarizes the recommendations from each of these tasks as developed in the body of this report. Specifically, Section 7.3 provides recommendations for follow-on research necessary to elaborate on the findings in this report.

1 Introduction

Safety-critical systems are developed by using rigorous system, software and airborne electronic hardware development assurance processes, following guidelines and standards such as SAE ARP4754A,[1] RTCA DO-178C,[2] and RTCA DO-254.[3] These assurance processes require explicitly specified behavior and traceability down to actual design (e.g. source code).

The aerospace industry is beginning to investigate how to apply artificial intelligence (AI) and extend development methods to implement new complex functions. The application of AI technologies such as machine learning (ML) requires new approaches to assuring overall system safety, due in part to the lack of explicitly specified functional behavior.

For example, the US Air Force Chief Scientist Report on "Technology Horizons: A Vision for Air Force Science & Technology During 2010-2030" summarizes the certification challenge with using artificial neural networks (ANNs), one type of ML algorithm, for highly adaptive autonomous systems:

> *"Developing certifiable verification and validation methods for highly adaptive autonomous systems is one of the major challenges facing the entire field of control science, and one that may require the larger part of a decade or more to develop a fundamental understanding of the underlying theoretical principles and various ways that these could be applied."* [4]

United States Code of Federal Regulations (CFR) Parts 23, 25, 27, and 29, all have an airworthiness standard that states, **"the equipment, systems, and installations must be designed and installed to ensure they perform their intended functions under all foreseeable operating conditions."** If machine learning is used, for example, to increase the level of flight control autonomy, these current standards imply the applicant must demonstrate the autonomous flight control function performs its intended function under all foreseeable operating conditions. The operating conditions can be infinite and may need to include rain, inflight icing, snow, dust, viewing angles, sensor imperfections, lighting changes, system failures, sensor degradation etc., all of which must be accounted for in order to show evidence of safe operation of a system including ML applications.

The scope and tasks of this AFE are intended to address the following fundamental certification questions:

1. What performance-based objectives do applicants need to satisfy to demonstrate a system featuring machine learning applications meets its intended function in all foreseeable operating conditions, recognizing that the foreseeable operating conditions may need to be bounded?
2. What are the methods for determining a training dataset is a) correct, and b) complete?
3. When is machine learning "retraining" needed and how is the extent of the retraining determined? How much retraining is required, for example, if changes are made to the sensors, neural net structure, or neural net activation functions, etc.?

---

1 SAE International, "Guidelines for Development of Civil Aircraft and Systems," ARP4754A, Dec 2010.

2 RTCA, "Software Considerations in Airborne Systems and Equipment Certification", Washington DC, DO-178C, Dec 2011.

3 RTCA, "Design Assurance Guidance for Airborne Electronic Hardware", Washington DC, DO-254, Apr 2000.

4 Dahm, W. J. A (Chief Scientist of the Air, Force Working Group Chair); AF/ST-TR-10-01-PR, Vol. 1, "Technology Horizons: A Vision for Air Force Science & Technology," 15 May 2010.

4. Can the simplex architecture monitors work for all machine learning applications, including complex functions such as ML classifiers? What are the conditions when a simplex architecture cannot work?

1.1 AFE 87 Project Description and Scope

The AFE 87 Machine Learning project focused on the safety and certification aspects of using machine learning techniques for certified air vehicle platforms. This project was divided into three major tasks:

Task 1 started with a literature search of Machine Learning methodologies to identify those most applicable to safety-critical aviation systems. The search intended to identify certification gaps in those approaches and included investigation of how adjacent industries, such as automotive, are approaching machine learning implementation and certification. The Task 1 deliverables are contained in this report, primarily in Section 2, which summarizes the current state of machine learning methodologies, adjacent industry approaches, and the certification gaps when applying these methodologies to aviation safety applications. A summary of the literature search results is given in Section 3.4.

Task 2 focused on the certification aspects of implementing a machine learning process flow by three use cases: using machine learning as a development tool only, machine learning used in a runtime monitoring environment, and dynamic machine learning that modifies runtime behavior in response to continued learning. During the research, the team decided to focus on ML design aid and dedicate its efforts on establishing the acceptability criteria on major characteristics such as: giving some requirements in terms of the dataset, robustness, adversarial inputs, observability and interpretability for aerospace applications. For this task, the primary question narrowed the focus and asked: What is a reasonable process flow for machine learning in applying this technique to certification of aircraft systems and what use cases are appropriate to use in assessing their value? The results from Task 2 are found in Section 3, Safety Considerations When Using Machine Learning and Section 4, Machine Learning Process Flow for Safety-Critical Applications.

Task 3 examined methods for bounding the runtime behavior of a complex system that employs machine learning computational techniques. This task investigated the viability of the different methods for implementing a “safety net” around the non-explicitly specified part of the system.

The results from this task addressed the questions:

- Are there one or more bounding approaches that can be used for complex systems by limiting their behavior through reversion to simpler systems?
- Are there schemes where it makes sense to set up such “safety nets” for certification purposes?

The Task 3 deliverables are contained in this report in Section 6, Considerations for Bounding the Behavior of ML Algorithms, including a discussion on the viability of the bounding methodologies and a set of certification guidelines for a bounding safety net.

1.2 Document Overview

In Section 2 we introduce Machine Learning by its definition, classifications, applicability, the current methodologies and the adjacent industry approaches. Section 3 highlights the challenges faced by practitioners attempting to use Machine Learning in safety-critical domains. Section 4 introduces a recommended process that should be followed for using Machine Learning in a safety-critical application. Section 5 provides guidance for preparing goals, arguments and evidence to support a safety analysis. Section 6 describes approaches to bound behavior including runtime assurance. Section 7 provides guidance for practitioners, standards authorities and for future research. Section 8 is a glossary of terms used in the paper, while Section 9 is a table of references used in this report. Finally, Appendix A provides an extended treatment on observability in machine Learning applications.

2 Machine Learning Current State

Machine learning models have shown impressive results in a wide range of problem domains including image classification,[5] object identification and tracking,[6] motion planning, anomaly detection and control (such as those present in self-driving cars),[7,8] and natural language processing.[9] However, these techniques are often black box in nature; particularly the recent developments in deep learning (DL) models.

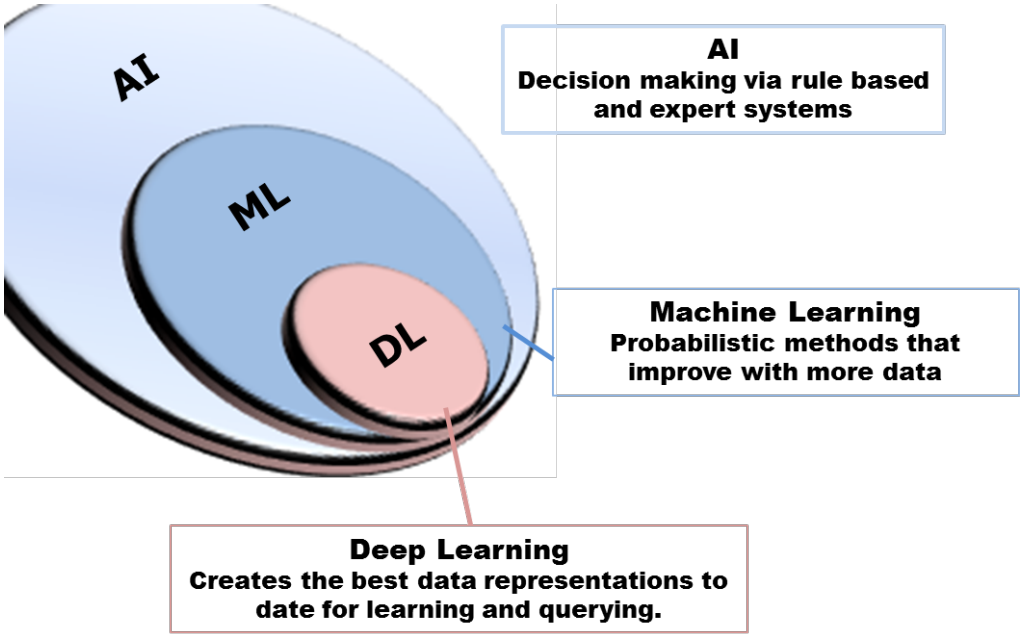

Figure 1: Artificial intelligence, machine learning and deep learning.

Figure 1 illustrates how machine learning and deep learning are nested subgroups of AI. The differences between ML and DL are often unclear to non-practitioners. Deep learning has superior performance to ML on a wide variety of tasks including speech recognition, natural language, vision, motion planning, and gaming. Though generalized, the primary points of comparison are listed in Table 1.

---

5 Xie, et al. "Aggregated Residual Transformations for Deep Neural Networks." ArXiv.org, 11 Apr. 2017, https://arxiv.org/abs/1611.05431.

6 Ren, et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." ArXiv.org, 6 Jan. 2016, https://arxiv.org/abs/1506.01497.

7 Bojarski, Mariusz; Testa, Davide Del; Dworakowski, Daniel; Firner, Bernhard; Flepp, Beat; Goyal, Prasoon et al. (2016): "End to End Learning for Self-Driving Cars." Available online at https://arxiv.org/pdf/1604.07316v1.

8 Qureshi, et al. "Motion Planning Networks." ArXiv.org, 24 Feb. 2019, https://arxiv.org/abs/1806.05767.

9 Vaswani, et al. "Attention Is All You Need." ArXiv.org, 6 Dec. 2017, https://arxiv.org/abs/1706.03762.

Table 1: Modeling aspects of machine and deep learning.

Modeling Aspects	ML	DL
Scaling	Model training typically plateaus more quickly.	Often continue to improve with more data.
Feature Engineering	Domain experts determine which information in the data (features) receive focus.	Minimal to no feature engineering. DL algorithms learn which information to emphasize through optimization without expert input.
Ease of transfer	Models are usually narrow in scope and less adaptable across domains.	Models often adapt seamlessly to multiple applications. E.g., publicly available pre-trained object recognition models form the basis of many computer vision applications.
Interpretability (XAI)	More interpretable due to explicit feature design and simpler models.	Explainability is primarily attained through heuristic probing of the model behavior after training.
Computation	Are relatively inexpensive and fast to train. Special hardware is less of a constraint.	Requires specialized hardware and large SSD and RAM memory. Often time intensive to train, but resource use decreases once training is completed.

Challenges in interpreting the underlying behaviors of trained ML models and the traceability to requirements (particularly from the perspective of current traceability practices) makes certification of safety-critical systems with embedded data-based AI unattainable using existing certification tools and processes. Interpretability, sometimes referred to as “explainable AI” (XAI), of DL systems is usually attained through heuristic probing methods after the model has been trained. Despite rapid progress on such methods to increase user trust of DL models, this remains an open challenge.

2.1 Machine Learning Overview and Perspective

Machine learning can be broadly classified according to type: Unsupervised, Supervised and Reinforcement Learning, as illustrated in Figure 2. Each of these techniques can be further combined, and often are, within a system. These learning types can be distinguished by factors such as the underlying learning mechanism, objective/cost function to be optimized and/or the supporting infrastructure. This section provides a high-level explanation of these different techniques.

2.1.1 Unsupervised Learning

Unsupervised learning (UL) is a common technique when a label (or target value) is not provided within the training dataset. This category of ML relies on clustering of data based on similarities in latent features of the datasets. For example, UL techniques could be used for anomaly detection on an aircraft by analyzing and clustering feeds from sensors distributed throughout the system, based on different modes of operation. Over time, an unsupervised model could learn to highlight differences between nominal and off-nominal sensor data. Such a detection model might be used to forecast and predict likely errors within the aircraft.

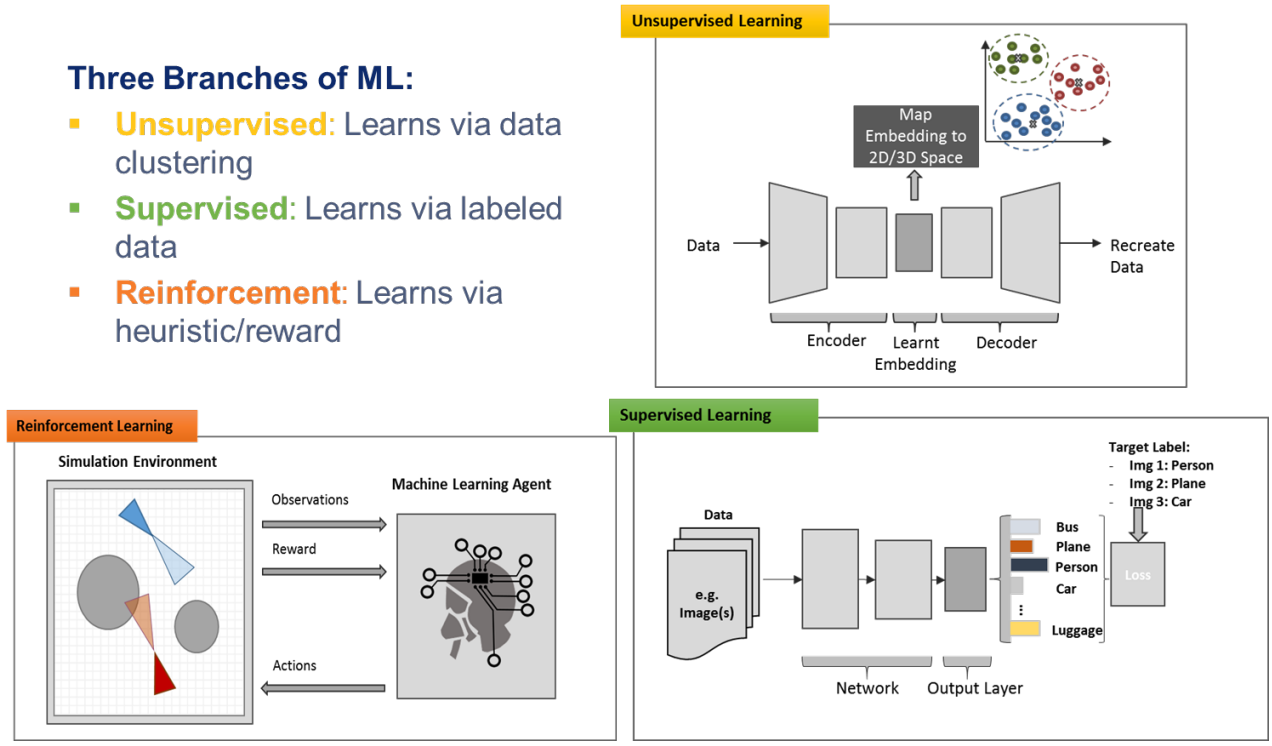

Figure 2: Three primary branches of machine learning: Unsupervised, Supervised and Reinforcement Learning.

2.1.2 Supervised Learning

Supervised learning (SL) is utilized when the underlying dataset has a target or label associated with each element of the set. Continuing with the predictive maintenance example, a SL system could learn when to predict system failures based on a dataset of prior flight logs, where each time-stamped example might be the time series of sensor readings or data from the health management system. The label could be a simple failure/no-failure or the exact type of failure detected. With training, a SL model will learn patterns in the input data (sensor feeds) indicative of a current or future failure. The model could then be deployed on a live system to help assist in predictive maintenance. Supervised learning models optimize their outputs by updating internal model parameters to minimize the difference between their predicted values and the expected value specified in the dataset.

2.1.3 Reinforcement Learning

Reinforcement learning (RL) is yet another ML paradigm that optimizes a model via a reward heuristic that indicates the quality of an action taken at each iterative step. In RL, an “agent” (i.e. the underlying model - also known as a policy) receives state information from the environment with which it interacts. This environment could be real or simulated. It is common to use a simulated environment for preliminary training and testing of an RL agent before deploying (and possibly refining) the model in a real-world setting. State information is then pre-processed as needed and passed into the RL model as input features. Input features are fed forward through the model, commonly a deep learning network, which manipulates the data via a series of layered weighted matrix operations to produce an output (known as an “action” in the RL paradigm). The action is then provided to the environment, which will provide a reward (rewards can be dense or sparse depending on the environment) and cause a transition to a new state by taking the action in the current state. The nature and value of the reward is

context and developer dependent. As an example, a reward to an RL-based autopilot system might be a positive value for maintaining steady, level flight at cruise altitude, and negative for deviations from the expected thresholds.

2.2 Deep Learning Overview

Black box systems hide or reduce human insight into the internal logic governing system output decisions (predictions, actions, controls, natural language processing, etc.). Deep learning has shown remarkable successes within a number of industries. Given the success of deep learning and its non-intuitive internal logic, this technique will be a focus throughout the document for understanding black box ML techniques in safety-critical aviation systems.

As described above, deep learning is a technique within the discipline of machine learning and the broader artificial intelligence domain. At a high level, deep learning differs from other machine learning techniques by learning both relevant features and their respective weighting (or influence toward a particular output). In deep learning, the raw input (time series of text, images, sensor data, etc.) is fed directly into an artificial neural network model. In other ML techniques, relevant features might first be extracted from the input based on expert or domain knowledge prior to being fed into the system. The ML system would then learn appropriate weighting for these extracted expert features. This is shown in Figure 3. In addition to built-in feature extraction, deep learning techniques have additional benefits such as constant computation time and memory usage, homogeneous structures that enable a more intuitive trade-off between resolution and execution time (i.e. less layers speeds up time to execute, but may reduce accuracy), and the ability to generalize to different problem sets.

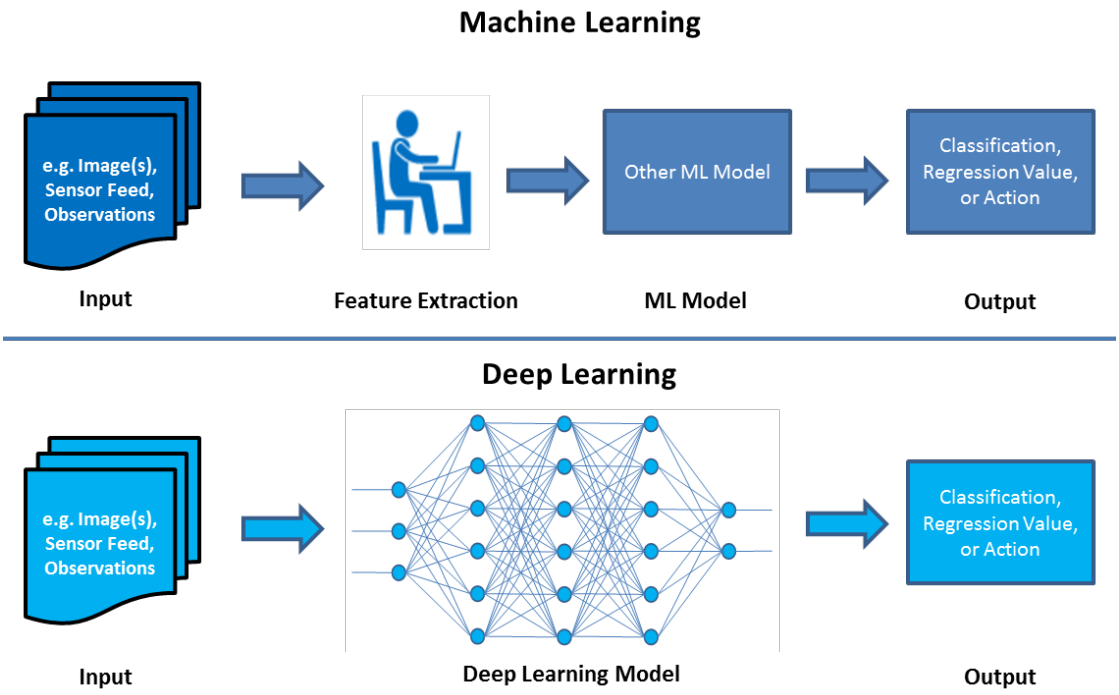

Figure 3: High level difference between machine learning and deep learning systems.

2.2.1 Artificial Neural Networks (High Level Perspective)

Artificial neural networks are computational algorithms inspired by simplified models of biological neurons found in brains. A basic ANN is shown in Figure 4. A typical ANN consists of an input layer, an output layer, and zero or more hidden layers. An ANN with two or more hidden layers is sometimes referred to as deep neural network (DNN). In this document, the term ANN is used generically, unless it is important in the context to distinguish between ANNs and DNNs. Each layer is composed of one or more nodes. Each node in a layer is connected to all the nodes in the previous layer using a weighting function. Within each node, the sum of all the weighted inputs is computed and then applied to an activation function. The activation function, which may include a bias, then determines the output of the node that is passed to the next layer.

For example, the ANN in Figure 4 consists of an input layer, three hidden layers, and an output layer. The input layer has three nodes, each hidden layer has 6 nodes, and the output layer has 2 nodes.

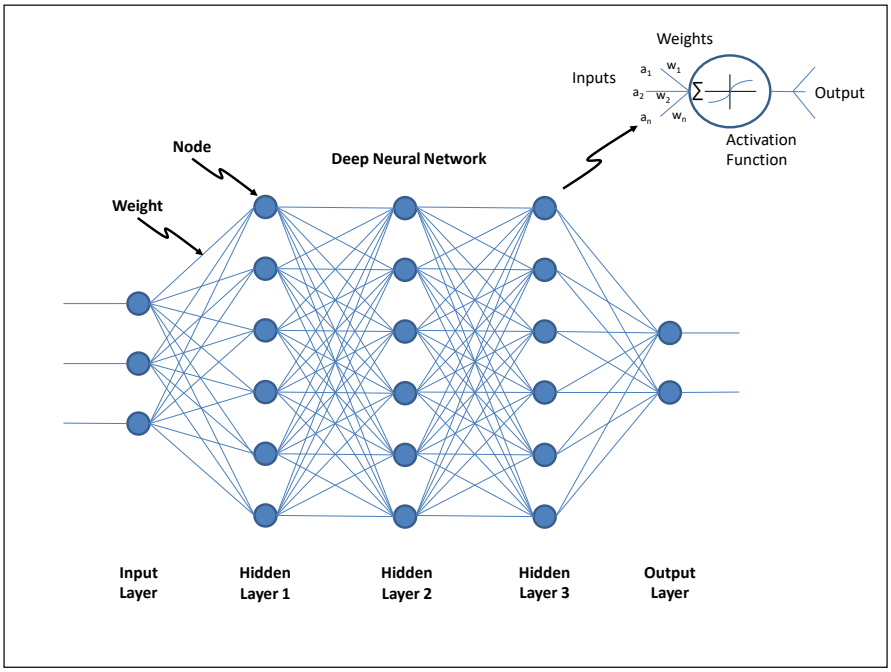

Figure 4: Schematic of an artificial neural network.

The ANN depicted in Figure 4 is a simple implementation but does illustrate the major topological features of more complex deep neural networks and provides a basis for describing the issues with using an ANN for safety-critical applications. The graphic above indicates what is inside the "black box" of an ANN. This example structure is considered a "feedforward network." Within a network implementation, there is an input layer (exposed to the outside to take in data) that feeds hidden layers eventually connected to the output layer. To train an ANN involves optimizing the values of the weights and activation functions of each connection and node in the neural net. The objective of training is to find values for these parameters to implement the desired ANN function and satisfy a given performance criterion. "Freezing" or monitoring the values of the nodes in the hidden layers at a particular moment can provide insight into the internal behavior of the black box system. However, due to the complexity of the learned task (number of variables and input scenarios), and the size of the deep learning network, concrete insights into the internal logic of the model can be rather difficult and unclear.

2.2.2 Artificial Neural Networks (Detailed Perspective)

The utility of deep learning is founded on the implementation of Neural Networks, as shown in Figure 3 and Figure 4. Neural networks are an approach for optimizing internal parameters (in this case network weights, similar to other adaptive system parameters) toward a preferable output, given a set of inputs. Unlike normal regression or classification techniques, Neural Network structures consist of multiple layers of "learnable" weights. As shown in Figure 4, this particular network has an input layer (the dimension of which, for example, might be the number of pixels in an image), several hidden layers, and finally an output layer predicting the value associated with the input (e.g., name, age, or gender associated with the input image).

Each of the layers are comprised of a number of nodes that perform a weighted summation of their respective inputs, and then feed that summed value through an "activation" function. As shown in the inset diagram, the inputs $a_1, a_2, \dots, a_n$ are multiplied by their respective weights $w_1, w_2, \dots, w_n$. The products of the inputs multiplied by their weights are then linearly summed. This summed value is passed to an activation function, of which there are numerous types.

Ultimately, the activation function bounds the summed value. For instance, a rectified linear unit ("ReLU") activation function might set all values less than 0 to a fixed value of 0, and all values greater than 0 are passed unaltered. Additional activation functions that have been implemented include sigmoid and hyperbolic tangent functions. Regardless of the specific form, activations functions provide the necessary non-linearity needed to adequately model real-world, nonlinear datasets, as well as enable more desirable convergence properties (i.e. avoiding undefined or vanishing gradients when updating the internal weights).

Essentially neural networks perform a series of weighted matrix multiplication operations followed by non-linear bounding/thresholding functions. For this reason, Graphics Processing Units (GPU), Tensor Processing Units (TPU) and other custom hardware that excel at performing calculations on large sets of matrices have enabled massive improvements in the speed of neural computation.

While it is currently difficult to fully expose the Neural Network black box (an active area of research as we will discuss later in this document), there are intuitive explanations for the internal logic of these models. It has been shown that the layers of neural networks learn to focus on increasingly more complex features within the input space.[10,11,12] For instance, the first layer might learn to detect edges or sharp contrasts between pixel values. Further layers might learn to detect more complex shapes, or individual features such as eyes, noses, and mouths in the case of image recognition. The final layers could then learn the appropriate relations between the individual features detected in prior layers (i.e. eyebrows are positioned above the eyes, noses are between mouths and eyes, etc.).

Once the data has been collected and the neural network topology developed, the network must be "trained" to optimize its predictions. In order to train neural networks, a large set of collected data points (representative of the problem domain) are fed through the internal weighted structure of the network topology to generate respective outputs. These outputs then go into a "loss function" (also known as cost function) to quantify how well the network performed based on expectations. Through gradient descent and back-propagation (not discussed in detail here), the weights of the network are updated

---

10 Zeiler, et al. "Visualizing and Understanding Convolutional Networks." ArXiv.org, 28 Nov. 2013, https://arxiv.org/abs/1311.2901.

11 Olah, Chris, et al. "The Building Blocks of Interpretability." Distill, 6 Mar 2018, https://distill.pub/2018/building-blocks/.

12 Erhan, Dumitru, et al. "Visualizing Higher-Layer Features of a Deep Network." (2009).

by minimizing the difference or quality of its output prediction relative to the expected output (or more advantageous output in the case of RL).

Trained deep neural networks often can be greatly reduced in size by a process known as pruning. Common variations of pruning remove weights and nodes of the network that contribute the least to the network outputs.[13] It is not uncommon to see pruning in excess of 90% of a network to minimally impact performance. This is especially useful for saving space and memory when embedding these models in electronics and mobile devices. From the assurance perspective, this process is analogous to removing "dead code" in software, which is intended to reduce the probability of unintended function.

2.3 Machine Learning Systems Development Cycle

The training-to-deployment pipeline for machine learning techniques is fairly consistent. This section aims to provide a common understanding of that pipeline (Figure 5). There are numerous means of mentally compartmentalizing the stack from data ingestion to deployment. However, each pipeline follows a similar set of processes: data intake and preprocessing, hyperparameter and configuration setting, model building, tuning, model training, model testing, and finally model deployment.

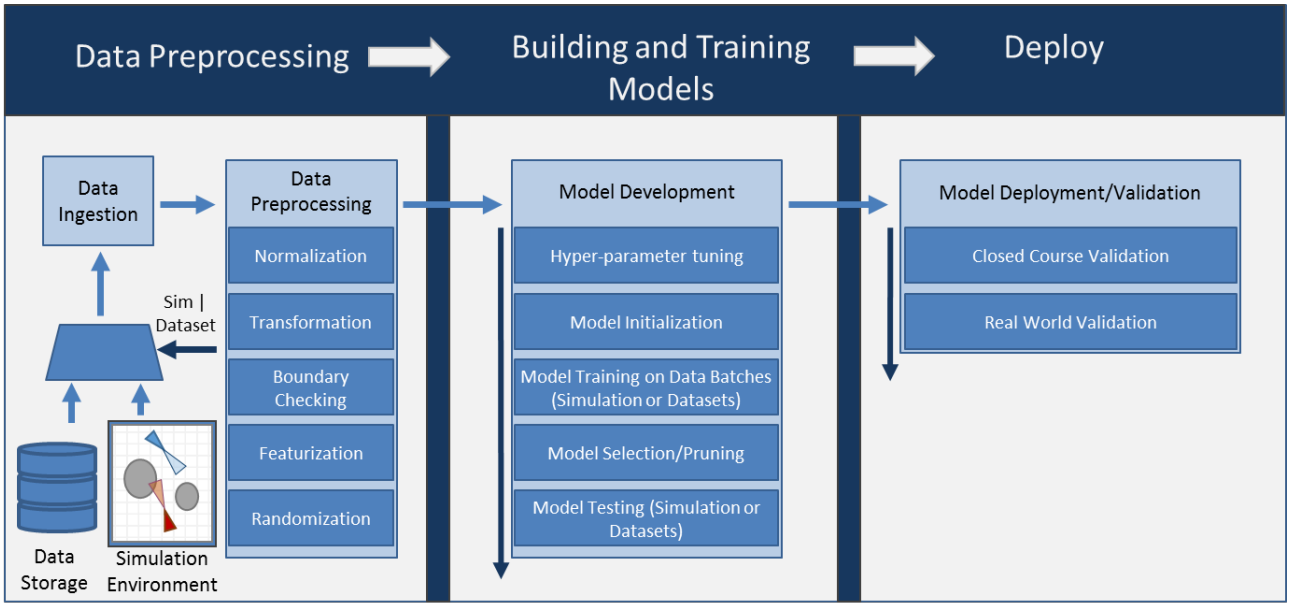

Figure 5: Common machine learning pipeline.

In the data intake and preprocessing step, datasets are retrieved from a centralized or decentralized database (or usually a simulation environment in the case of Reinforcement Learning). The data is then preprocessed (or "cleaned") before being provided to a machine learning model. Preprocessing data might entail removing noise, normalizing the data distribution, extracting relevant features if desired, and/or augmenting the original data (changing lighting, modifying orientation, masking portions of the data) as an attempt to generate more training data or facilitate greater generalization of the network.

Hyperparameters are chosen to configure the Neural Network structure and operate within the overall algorithm. Common hyperparameters could include the number of hidden layers, number of nodes,

[13] Cheng, Yu, et al. "A survey of model compression and acceleration for deep neural networks." arXiv.org, https://arxiv.org/abs/1710.09282v5 (2017).

learning rate (which controls the step-size in gradient descent), number of epochs (number of times to cycle over the datasets) and various other parameters governing the optimization techniques (Adam, Moment based gradient descent, etc.). For the purposes of this document, it is not important to provide a detailed discussion of the different hyperparameters, of which there can be many. Additionally, there are configuration variables relevant to defining the simulated environment or scenario, particularly in the case of RL. In order to expose an RL agent to a variety of situations, it is common to randomize the initial conditions of a simulated environment. For example, in the case of a Neural Network that controls an aircraft taxiing at an airport, the number of aircraft or personnel, positions, speed, and heading of these objects might be randomized or fuzzified within a certain minimum and maximum range. Additionally, parameters defining the type of aircraft, weather, time of day, and the specific airport or terminal could also be configured.

Model building or model selection can either be a manual or automated process in which different neural network topologies are generated. These network topologies can vary in a variety of ways such as input/output dimensions, the number of hidden layers, number of nodes in each of the layers, activation functions after each layer, normalization techniques (dropout,[14] batch normalization[15]) in between each layer, and network type (convolution, recurrent, fully-connected). Some of these parameters defining the network topology were likely provided in the hyperparameter tuning step. It is not uncommon to develop multiple neural networks for a given use case and choose a single or ensemble of the best performing networks.

Once the preprocessing steps have been completed, hyperparameters defined, and model(s) developed, the actual training-to-testing processes can occur. (Note, the description in this section leans toward supervised learning processes). To begin, the data is usually split into separate training, validation and testing sets.[16] The training set is used to update the neural network weights over a number of epochs (training cycles) toward the expected output. The validation set (not always included, but preferable in the case of supervised learning) is used to select the best subset of models before explicitly testing. The validation set (note: “validation” dataset is not to be confused with “validation” in the traditional systems engineering context of “V&V”) aids in encouraging more generalized models that help mitigate the bias-versus-variance tradeoff (i.e. a biased model is one that does not fit the training data well, while a high variance model fits the data too well). Finally, the testing dataset is fed into the network in order to evaluate the quality of the model.

---

14 Srivastava, N., et al., “Dropout: a simple way to prevent neural networks from overfitting,” Journal of Machine Learning, Res. 15, 1 (January 2014), pp. 1929-1958.

15 Ioffey, S. and Szegedy, C., “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” ArXiv.org, 2 Mar. 2015, https://arxiv.org/abs/1502.03167.

16 Kohavi R., “A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection,” Proceedings of International Joint Conference on AI, 1995, pp. 1137–1145. Retrieved from https://www.ijcai.org/Proceedings/95-2/Papers/016.pdf on July 31, 2019.

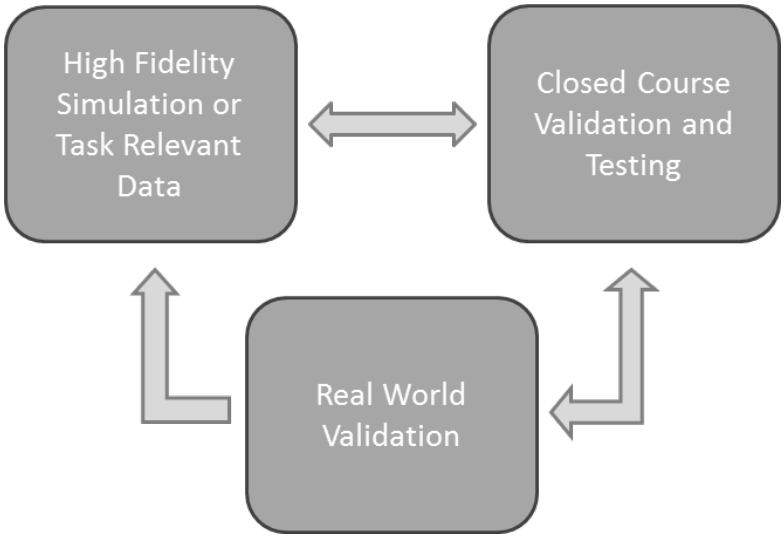

Figure 6: Automotive industry machine learning pipeline cycle.

A model will be deployed in a more real-world setting once it has been adequately tested in simulation or on the available training set. Figure 6 illustrates the intermediate step performed by many system designers, especially those in the automotive industry, who deploy their trained models in a closed-course environment, where the test conditions can be appropriately controlled (see safety reports from Uber,[17] Waymo,[18] Ford,[19] and General Motors[20]). Furthermore, closed-course environments enable the generation of new testing scenarios (and thereby more data for a ML model to ingest). Such closed-course testing is conceptually similar to aerospace flight testing, though flight testing typically has higher costs and complexity. Finally, the system (incorporating a machine learning model), is deployed in a real-world environment. Performance is monitored and operational characteristics are recorded. Often the cycle between training and testing is repeated in a cyclic fashion to continually refine the system design.[19] It is also important to note that the system should only be deployed in environments in which appropriate and adequate testing has been conducted. This is known as the Operational Design Domain (ODD) in the automotive industry (Figure 7). In this case, a vehicle is only allowed to operate in its ODD, for which conditions have been appropriately tested and ensured to provide safe operation. The operational domain accounts for factors such as geography, road types, speeds, weather, and laws and regulations. This concept could easily be extended to the aviation environment.

---

17 Uber Advanced Technologies Group: A Principled Approach to Safety, 2018. Retrieved from https://www.readkong.com/page/2018-uber-advanced-technologies-group-a-principled-approach-5983176?p=1 on June 19, 2019.

18 Waymo (2018): "Waymo Safety Report. On the Road to Fully-Safe Driving," 2018. https://storage.googleapis.com/sdc-prod/v1/safety-report/Safety%20Report%202018.pdf. Retrieved on June 19, 2019.

19 Markaby, S., A Matter of Trust: Ford's Approach to Developing Self-Driving Vehicles," 2018. https://medium.com/self-driven/a-matter-of-trust-fords-approach-to-developing-self-driving-vehicles-12f602887822. Retrieved on June 19, 2019.

20 General Motors, "2018 Self-Driving Safety Report," http://autocaat.org/webforms/ResourceDetail.aspx?id=4823. Retrieved on June 19, 2019.

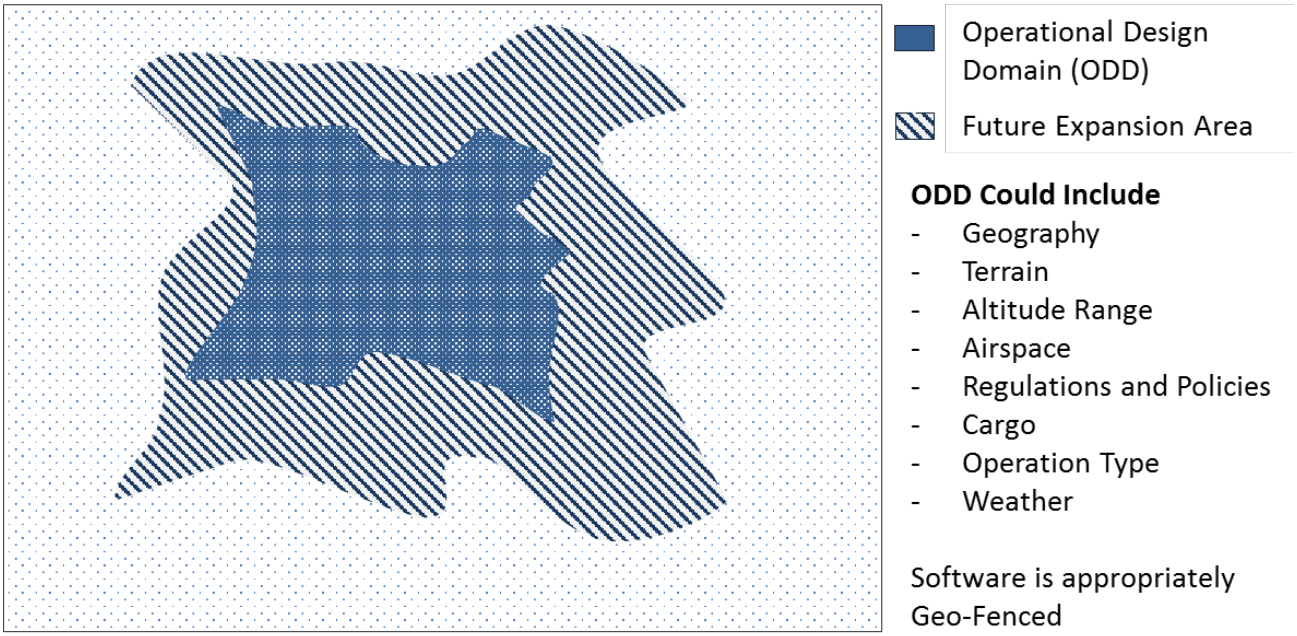

Figure 7: Automotive industry's concept of operational design domain.[18]

As a concluding remark, the pipeline above has subtle differences depending on the machine learning technique employed (Reinforcement Learning, Supervised Learning, or Unsupervised Learning). For instance, UL and SL typically use a pre-stored dataset, whereas RL often employs a simulated environment. The RL "datasets" are provided via the simulated environment. The concepts of data preprocessing, hyper-parameter tuning, model selection and stages of training to deployment still apply, however, the mechanisms for parallelizing the training of ML models and acquiring data has some differences.

2.4 Machine Learning Related Complexity Analysis

Machine learning techniques have been applied to increasingly complex environments. Likewise, the prerequisite computational resources necessary for state-of-the-art performance in these complex problem domains have also risen exponentially.

Artificial Intelligence is now used to compete and perform in a range of activities from human dominated strategy games (Chess, Go, Starcraft II) to vehicle control and autonomy (self-driving cars). The complexity of these environments can be analyzed through various metrics such as the rate of data generation or the search space of the environment. The search space can be envisioned as a tree-based graph structure where each node in the tree defines a new state (e.g. the board state in chess or the state of the vehicle in a fixed frame of the environment), and the edges connecting each node are one of the available actions taken to generate a new state within the environment. More formally, the search space can be calculated via the breadth (number of actions at each interaction point), and depth (the length of the game or scenario). As an example, the typical total length of a car trip per day might be 1.5 hours. The search space for Chess is 10^{123}, while that for Go is 10^{360}.[21] When learning in domains with discrete action spaces, the derivation of complexity is more straightforward, as there is a finite set of actions at each time-step of interaction. However, in environments where the action space

[21] Silver, D., et al. "Mastering the Game of Go without Human Knowledge." Nature News, Nature Publishing Group, 18 Oct. 2017, www.nature.com/articles/nature24270. Retrieved June 19, 2019.

is inherently continuous or real valued, the search space needs to be approximated by discretizing the available action space. For instance, the speed and steering angle of an autonomous vehicle has a range of continuous values applicable at each time step. In such a case, it might be helpful to think of complexity as the amount of data generated over a fixed period of time by the various onboard sensors or the amount of computational resources needed for development. For autonomous self-driving cars, roughly 4 terabytes worth of data are generated every 1.5 hours.[22] For modern machine learning systems, the amount of computational and data requirements is of the same scale.

The projected throughput necessary for modern ML systems can be greater than 1000 petaflops per second for one day (a petaflop is 10^{15} floating point operations per second).[23] It is common that trained ML models require an extensive network of interconnected computational resources, as the data generation and/or training process is often parallelized to enable optimization of multiple agents simultaneously. This is done to explore more of the environment's multi-faceted search space or develop multiple ML strategies that have unique objectives.[24] It should be noted that once the network is trained and the weights fixed, the computational resources needed may be greatly reduced.

2.5 Machine Learning Technical and Industry Trends

To conclude the high-level overview on the current state of machine learning and discussion into prerequisite ML knowledge, it is important to understand the current trends within AI and more specifically ML. The graphic in Figure 8 highlights technical trends.

The generation of software has become increasingly automated. The graph in Figure 8 highlights trends in computer vision software development. However, such trends in software development are not unique to computer vision. Historically (and still widely prevalent today) software was hard-coded (i.e. fixed rule-based logic is used as the driving behavior of software processes). Increasingly, software was (and is being) developed that learned its own internal weighting of relevant, but still hardcoded features. Fast-forward to more recent times: software is not only learning the relevant features given a set of inputs (e.g. a convolutional neural network given a set of inputs learns to decipher shapes, noses, mouths, entire faces automatically), but is also constructing the model topology. A number of companies are offering software as a service (SaaS) solutions to enable a broader non-technical audience the capability of developing trained ML models through automated processes (see Google AutoML for example[25]). Through this service, end users can provide their own domain specific data, and the system will output models best suited/optimized for the particular data given by learning to optimize a model's internal logic (model weights) as well as the model's structure/topology (number of nodes, layer size, layer types: convolutional, fully-connected, recurrent, etc.). If the trend continues, a number of industries will utilize artificially intelligent software to further refine and develop new task-specific AI software.

---

22 Geonovum and Geospatial. "Self-Driving Vehicles [SDVS] & Geo-Information," https://geospatialmedia.net/autonomous-vehicles-geospatial-report.html?utm_source=press-release&utm_medium=referral&utm_campaign=july2017. Retrieved June 19, 2019.

23 Amodei, Dario. "AI and Compute." OpenAI, OpenAI, 16 May 2018, https://openai.com/blog/ai-and-compute/. Retrieved June 19, 2019.

24 "AlphaStar: Mastering the Real-Time Strategy Game StarCraft II." DeepMind, https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/. Retrieved June 18, 2019.

25 "Cloud AutoML - Custom Machine Learning Models | Google Cloud," Google, https://cloud.google.com/automl/. Retrieved June 18, 2019.

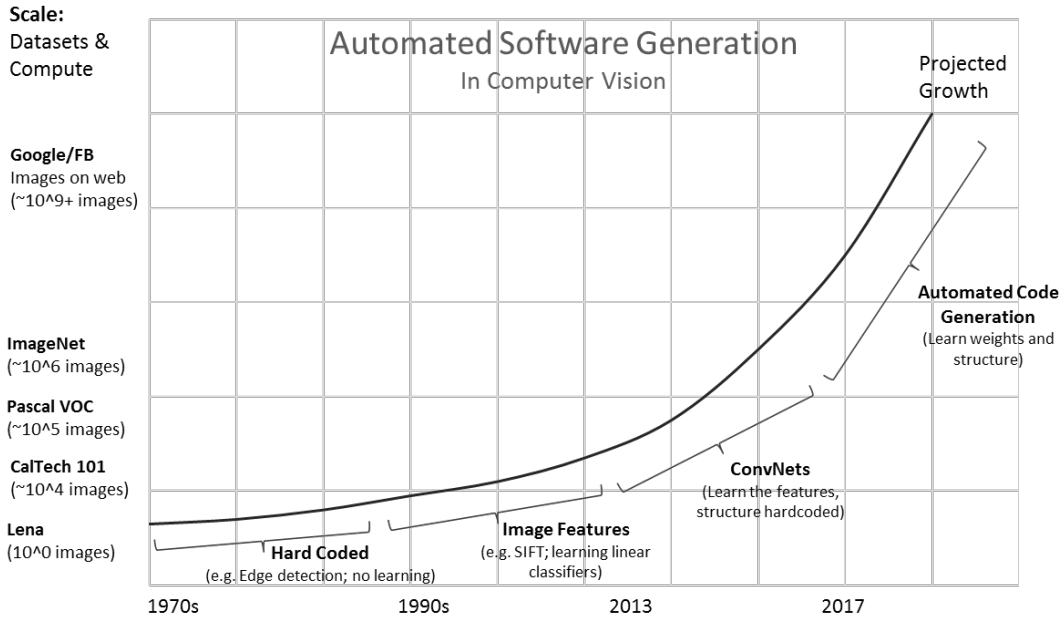

Figure 8: Automated software generation in computer vision.[26]

2.6 Certification Challenges When Using ML in Safety-Critical Applications

The advantages of AI/ML coupled with the rapid advances in the development and deployment of ML as described above have led to an immediate desire to incorporate ML technology in certifiable aerospace systems. Aviation certification authorities and applicants have grown interested in understanding how ML systems can be assured as safe for flight given that they differ fundamentally from traditional systems for which requirements are explicitly traceable to specific software and hardware design elements. Tools and processes developed to assure existing systems do not necessarily extend to data-defined systems such as those incorporating ML. This has led to a number of questions and observations that must be addressed by the aerospace industry (and which led to the formation of this project):

1. Industry standard development assurance processes such as RTCA DO-178C, RTCA DO-254 do not have guidance for ML algorithms. If machine learning is used, it may not be possible to satisfy all RTCA DO-178C and RTCA DO-254 objectives such as those associated with the low-level requirements, implementation, integration, and verification activities. The main reason for this is the lack of a meaningful representation of the internal structure of the machine learning implementation. For example, while it may be possible to extract and represent the weights and connections between the neurons of an ANN, the information does not describe the intended behavior.
2. Some machine learning implementations may exhibit adaptive behavior, which may impact traditional notions of repeatability. Even in implementations that are non-adaptive, meaning the implementation is fixed after training, claims for repeatability may not be trusted since the internal states may not be fully understood.

[26] Karpathy, A., “TRAIN AI 2018 - Building the Software 2.0 Stack.” Vimeo, 10 May 2018. http://vimeo.com/272696002. Retrieved on June 19, 2019.

3. Machine learning implementations require extensive data for training. For a subset of flight tasks, training data could be developed. Because of the infinite possibilities for routine, non-routine and emergency conditions, development of extensive training data for these conditions may not be feasible. Therefore, the operation of the machine learning implementation may need to be bounded based on the scope of the training and verification.
4. Training machine learning implementations may be performed using datasets from live, or historical interactions. The data may also be provided by simulations that mimic the actual behavior of the physical system. If live or historical data is used, then the data should be a true representation of the system behavior over its entire range of foreseeable operational conditions. If simulators are used, then the simulator should be qualified to appropriately model system behavior throughout the range of foreseeable operational conditions.
5. Some machine learning algorithms process sensor data such as radar, light detection and ranging (LIDAR), and high-resolution cameras to detect and identify objects to support flight automation. The flight automation system using these sensors could have degraded performance under different conditions such as snow, inflight-icing, rain, fog, smoke, different light conditions, and different viewing angles which could negatively affect their intended functionality of the flight automation system relying on these sensors. Also, small imperfections in the sensor (e.g. scratched lens) or a sensor that is not calibrated could negatively affect their performance. Sensors may also degrade over time.
6. It may be difficult to demonstrate that a machine learning implementation does not have any unintended functionality. Existing verification methods such as traceability and coverage analysis cannot be used to demonstrate that the machine learning implementation does not contain unintended functionality. For example, in an artificial neural net implementation, the weights and interconnects may not be analyzable, making it impossible to perform a reverse trace from the ANN implementation to its requirements.
7. Machine learning implementations are trained over a finite set of inputs and outputs and are expected to generalize, i.e. to behave correctly for previously unseen inputs. However, machine learning implementations can react in unexpected and incorrect ways to even slight perturbations of their inputs. This unexpected behavior of machine learning implementations can result in unsafe systems or restrict the usage of machine learning implementations in safety-critical applications. Hence, there is a need for methods that can provide formal guarantees about ML implementation behavior. Unfortunately, manual reasoning about large ML implementations is impossible, as their structure renders them incomprehensible to humans. Therefore, automatic verification techniques are needed, but here, the state of the art is a limiting factor. There exist approaches for finding adversarial inputs, but the ability to verify their absence is limited.[27]
8. Machine learning implementations can periodically fail to perform their intended function due a high functional error rate*. The automotive industry is facing the same challenge of how to safely introduce ML implementations for automobile driving automation. Salay, Queiroz, and Czarnecki state, “a machine learning model typically does not operate perfectly and exhibits some error

[27] Katz, G., et al., “Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks,” Elsevier Annual Reviews in Control, Volume 34, Issue 1, April 2010, pp. 163-174. https://arxiv.org/abs/1702.01135, Retrieved June 19, 2019.

rate. Thus, correctness of a machine learning component, even with respect to test data, is seldom achieved and it must be assumed that it will periodically fail."[28]

*Note: An ML system has "failed" if it exhibits a systemic error manifesting as non-repeatability of input to output. These are probabilistic variations and not random occurrences.

[28] Salay, R., Querioz, R., and Czarnecki, K., "An Analysis of ISO 26262: Using Machine Learning Safely in Automotive Software," www.dtic.mil/dtic/tr/fulltext/u2/1003182.pdf, Retrieved May 16, 2019.

3 Safety Considerations When Using Machine Learning

Safety is of primary importance in the aerospace domain. Traditional aerospace processes such as SAE ARP4754A, SAE ARP4761,[29] RTCA DO-178C, and RTCA DO-254 form the current accepted basis for safety assurance processes. Application of ML techniques in safety-critical systems pose a series of challenges to accepted processes. AI researchers identify a series of failure modes that may be related to using ML. Safety considerations are analyzed in order to "mitigate the accident risk".[30] Moreover, accidents can be defined as "unintended or harmful behavior that may emerge from any systems (e.g. a software) when we specify the wrong objective function, are not careful about the development process, or commit other software-related implementation errors."[30] As applied to safety-critical aerospace systems, these considerations become even more important as automation and autonomy implement complex tasks.

The application of machine learning systems to safety-critical areas provides new and additional challenges to safety engineering, such as an increase in the uncertainty that correct predictions and subsequent actions will be made, due in part to the nature of most ML algorithms outputs. Safe use of machine learning depends on the ability to assure that systems employing ML capabilities behave as intended or that unintentional risks and harmful behaviors are mitigated. Based on the observations and concerns presented by Amodei, et al.,[30] Faria,[31] and Dawson,[32] the list below itemizes some of the possible failure modes of ML systems:

- **Specification problems:** The incorrect specification of the formal objective function, where the solution designed and deployed by a human optimizes an objective function that results in harmful and unintended results.
- **Robustness problems:** Instances where a solution may have been specified by the correct objective function, but problems occur due to poorly curated training data or an insufficiently expressive model.
- **Oversight problems:** Instances in complex environments where feedback to assist a solution to achieve its objective function is expensive or computationally inefficient. Scalable oversight: ML system performs poorly because of the limited feedback input available.
- **Adverse side effects:** ML system disturbs the environment in negative ways while performing the designed function.
- **Unsafe evolution:** ML system does something harmful during its evolution, while it adjusts its parameters based on external feedback.
- **Distributional shift:** ML system does not recognize an environment different from its training environment or dataset.

The failure modes mentioned above derive from limited access to the actual specified function, insufficient or poorly curated training data, or an insufficiently expressive model and/or deficiencies in

---

29 SAE International, "Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment", ARP4761, Dec 1996.

30 Amodei, D., et al., "Concrete Problems in AI Safety," https://arxiv.org/pdf/1606.06565.pdf. Retrieved May 26, 2019.

31 Faria, J., "Non-Determinism and Failure Modes in Machine Learning," https://ieeexplore.ieee.org/document/8109300/. Retrieved May 26, 2019.

32 Dawson, N., "AI Safety Literature Review," (2017) https://bitsandatoms.co/ai-safety-literature-review/. Retrieved May 26, 2019.

part of the learning process. The ML system designer could mitigate unsafe evolution by taking advantage of risk-sensitive performance criteria; by expanding the simulated scenarios considered during training or retraining; by bound exploration for a new set of parameter values constraining outputs to certain regions of state space known as safe, so that the ML system can recover to safe or bounded behavior; by separating safety and performance function and obeying constraints on the safety function with high probability.

Additionally, in order to build safe and predictable systems it is critical to employ safety checks to detect such failures, and ultimately have statistical assurances about how often they happen. Other approaches focus only on modeling the distribution of errors of a model and training on multiple distributions. From this, it is also possible to evaluate when a set of inputs is too novel so that a reasonable performance cannot be expected. Those practices theoretically would support designing models that could bound their performance on new verification dataset distribution.

The ideal outcome of these approaches to limit side effects would be to prevent or at least bound the incidental harm an ML system can cause. Useful strategies to bound side effects would certainly not be a replacement for extensive testing or careful consideration by designers of the individual failure modes of each deployed system.

Formal rewards or objective functions are an attempt to capture the intended behavior, but it has been shown that ML systems can output unintended behavior even when the objective function is correctly optimized.[33,34] There are several reasons for this problem including:

- **Partially observed goals:** it is assumed for ML systems that reward is directly experienced, even if some aspects of the environment are only partially observed. Tasks often involve representing the world into some objective state, through imperfect perceptions and those imperfections lead to maximizing rewards without improving intended performance.
- **Complicated systems:** the probability that there is a viable hack affecting the reward function also increases significantly with the complexity of the ML system and its available strategies.
- **Abstract reward:** abstract concepts are captured by sophisticated reward functions; such abstraction demand makes it vulnerable to adversarial counterexamples.
- **Feedback loops:** when an objective function comprises a self-amplifying component that distorts specified intended behavior.
- **Environmental embedding:** in the formalism of reinforcement learning, rewards are considered to come from the environment. In addition to the risk that environmental embedding may present safety concerns due to a lack of robustness, it may also create cybersecurity concerns due to intentional manipulation of the embedded environment.

Traditionally, embedded software that has been applied to safety-critical areas has required near-full predictability of behaviors under all conditions, a detailed design with a rigorously specific set of

---

33 Thompson, A., “Artificial Evolution in the Physical World (1997),” from **Intelligent Robots to Artificial Life (ER’97)**, T.Gomi (Ed.), pp. 101-125. AAI Books, 1997. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.26.6187, retrieved July 31, 2019.

34 Bird, J. and Laysell, P., ““The Evolved Radio and Its Implications for Modelling the Evolution of Novel Sensors,” http://arxiv.org/pdf/1704.07911v1, retrieved June 19, 2019.

requirements, and a comprehensive set of verification activities to confirm the software implementation fulfils the specification.[35]

It should be noted that the need for formalized safety assurance has been recognized for decades, and SAE International has developed relevant Aerospace Recommended Practices (ARP's). SAE ARP4754, *Guidelines for Development of Civil Aircraft and Systems*, was published in 1996 (currently updated to revision 4754A). Its tightly-coupled sibling is SAE ARP4761, *Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*. While these and additional guidelines have been created, including RTCA DO-178C for software development and RTCA DO-254 for hardware development, new guidance must be created in order to guide the development life cycle for systems incorporating ML. These new standards will need to be incorporated in the suite of guidance material shown in Figure 9 to support future certification of aerospace systems incorporating ML.

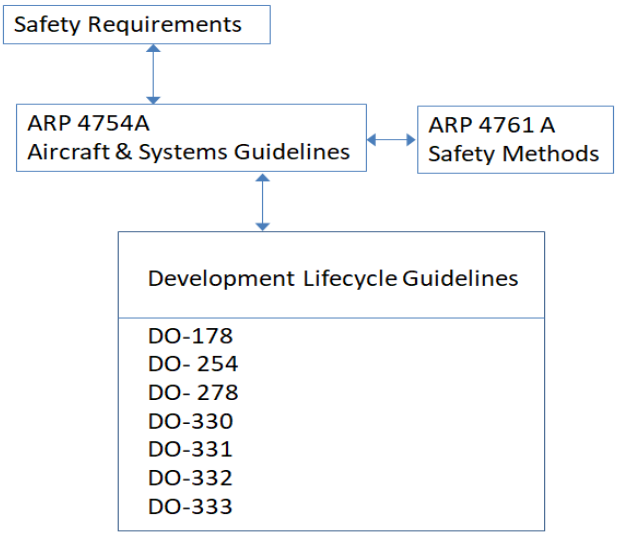

Figure 9: Safety assurance guidance.

To address the gaps in available guidance, elements of the ML system development life cycle must be analyzed in the context of the considerations described above. The following sections employ available literature and project members' expertise to identify gaps in knowledge or guidance that impact the aerospace industry's ability to adequate assure the safety of systems incorporating ML technologies.

3.1 Datasets

The first consideration for a "curated" dataset is about the "quality" of its measures. However, the definitions for current data curation standards are related to data organization, integration, and life cycle management. Additionally, the data curator role is more concerned with maintaining and managing metadata; including involvement in determining the best practices for working with that data and

[35] Dawson, N., "AI Safety Literature Review Revisited," (2018) https://bitsandatoms.co/ai-safety-literature-review-revisited/. Retrieved June 18, 2019.

supporting decisions on which data asset is more appropriate for use within a machine learning model.[36]

Below is a list of factors that influence data quality for ML solutions, according to Géron:[37]

- **Representativeness:** Training data should contain all foreseen scenarios in which the system will be used; training data should contain a representative number of rare examples; features in the dataset should be selected or generated from raw features according to their relevance to the function that the ML model will perform; in the case of unsupervised ML algorithms it should be checked if the information that is expected to be extracted from the data has been encoded.
- **Labeling:** In supervised learning, algorithms learn from a dataset of examples labeled with an output variable representing the right answer. Thus, the data should be collected and preserved in a way that avoids corruption of the labels. Each example in the training dataset should be checked to assure that it has an associated output (label).
- **Sufficiency:** Data should be sufficient in quantity (statistical significance), which will depend on the problem domain. In general, the more good-quality data, the better; but processing more data comes with a price of computational burden. In order to assure data quality, a data curation process is a good practice, i.e., to perform data organization, integration, management through its life cycle in order to make it useful for data-driven uses such as ML applications.

The majority of datasets contain bias, noise, constraints, outliers and missing values. That is the reason for a designer/developer of the solution to execute some steps in order to guarantee the quality of a curated dataset. Some of the data preparation activities that are part of machine learning model development are:

- data cleansing aimed at removing irrelevant information;
- data transformation as part of formatting, completing data properly and generating new features from the raw data;
- data augmentation to get a larger dataset, (this should be done cautiously to avoid injecting uncertainty to the dataset);
- verification that data inputs are in the expected format;
- verification that data inputs are free from discrepancies in values or missing data;
- verification of data inputs for dependency on other inputs;
- verification that data inputs are actually available during system operation;

Data quality assurance should be part of a data-driven model design to specify a set of high-level requirements to accomplish the desired system behaviors. After this, low-level and derived requirements relating to dataset pre-processing and post-processing and to model implementation can be specified.

Finally, it is expected that during design and development, designers will select appropriate models and employ the best datasets available to develop a reliable ML-based system that reliably exhibits its

---

[36] Miller, R., “Big Data Curation,” 20th International Conference on Management of Data (COMAD), Hyderabad, India, December 17-19, 2014, http://comad.in/comad2014/Proceedings/Keynote2.pdf, retrieved on Feb. 7, 2019.

[37] Géron, A., Hands on Machine Learning with Scikit-Learn and TensorFlow, O’Reilly Media, March 2017, http://shop.oreilly.com/product/0636920052289.do.

intended behavior. However, implementations should also provide an ability to detect inputs that result in undesired behavior during runtime and identify these inputs as new requirements or constraints. In other words, the input must be included in the dataset for future retraining to ensure that the dataset grows more representative over time, thus ensuring the ML model behaves in foreseeable conditions.

3.2 Adversarial Inputs and Robustness

The existence of adversarial examples raises a more general concern: it indicates that the ANN might be responding to completely different input features than do humans, and that rather than constructing human-like generalizations from the training data, it has merely memorized exactly what it has seen. Although adversarial examples are just that, constructed by “adversaries,” they raise the concern that classifications learned by ANNs may not be robust, and that naturally arising inputs different from the training set, yet close to it, may be classified incorrectly. Since we do not completely know how ANNs work, we cannot currently build a fully principled defense against this (or, dually, verification of its absence), but there are some promising ideas.

Practical application of machine learning in aerospace systems requires an understanding of how to quantify and ultimately predict bounds on the behavior of such systems when subject to inputs that were not considered during the training of the learning system. Quantifying the robustness of these systems, which we generally define here as the tolerance of system outputs to variations in the inputs, over the range foreseeable operating conditions, is central to system performance and, in some systems, may affect system safety.

We can consider at least two main types of sources for these variations: inputs that are intentionally modified, for example by an adversarial actor, with the purpose of misleading the decision-making algorithm; and variations in inputs that arise naturally in the course of system operation, not due to any intentional actions. Among the latter unintentional variations, we may further distinguish minor variations due to sensor noise, major variations such as might occur in the case of degraded or failed sensors, and variations due to inputs arising due to new operational conditions that were not well represented in the original training, verification or validation datasets.

Robustness of a machine learning system can be defined as the ability to identify the features that were encountered during training, even in a noisy environment. Regardless of the source of the input data variation, the robustness of the system depends on the ability of the learning system to generalize features that were encountered during the training of the system, so that the modified, degraded or unexpected inputs are processed appropriately. In case of unintentional data variations, noise or degradation models may help constrain the input state space and/or be incorporated in training. In the case of adversarial inputs, the targeted system may need to handle inputs that are intentionally varied with a goal of causing unintended system behavior. This presents unique challenges to designing robustness into the learning system that have led to an active subfield known as Adversarial Machine Learning. This subfield addresses both “deliberate” attempts to adversely affect the machine learning algorithms, as well as “natural” phenomena such as sensor degradation and environment wear/aging of the natural environment.

There is a growing body of literature in Adversarial Machine Learning that considers both the analysis of vulnerabilities in learning algorithms and techniques to make learning more robust. Adversarial attacks may be classified by the point in the system life cycle at which the attack takes place: “evasion attacks” occur after a system is deployed, while “poisoning attacks” are attempts to corrupt the learning system by manipulating training data.

An example of an evasion attack may be an attempt of a spam email generator to avoid detection by intentionally changing features of the email content and monitoring the success rate of defeating spam filters. Such attacks can be performed with or without the knowledge of the specific training algorithm

that is used by the machine learning system ("white box" or "black box" respectively). Since these types of attacks could potentially affect aerospace systems that were trained prior to certification and deployment, design assurance processes must account for these types of attacks. Some new techniques have been emerging that may improve robustness with respect to evasion attacks. The literature suggests that white box attacks are potentially more dangerous due to the attacker's access to more information about the system. This should lead to considerations about the usage of open-source software to implement machine learning. Unrestricted or publicly available software could open safety breaches in a system classifier for instance with potentially severe consequences on the safety. This consideration extends to the knowledge of training datasets and training methods. While incorporation of third-party content in aerospace systems is not unique to machine learning, there are unique characteristics that must be incorporated in engineering processes used to assure third-party content. Persistence of training data and the inherent obscurity of the machine learning architectures make it essential that these assurance processes be adapted to ensure that malicious content cannot create system failures.

Poisoning attacks generally require access to the training dataset in order to manipulate the learning system behavior. In this regard, good cybersecurity controls during system development are necessary to ensure the integrity of the training dataset, as elucidated in the previous section. However, systems that may employ unsupervised learning, or learn from actual environments encountered during operations, might be more susceptible to poisoning and need to have additional requirements to ensure continued airworthiness. From this standpoint, systems that allow continual learning and adaptation after deployment may be seen as more susceptible to adversarial attacks, while systems that are "frozen" after deployment are naturally more robust. Note that if such a "frozen" deployed system is periodically retrained with a new operational dataset, the re-trained model may become sensitive to adversarial attack.

The very existence of adversarial examples raises a more general concern: it indicates that a machine learning algorithm might be responding to input features that are completely different from those that are recognized by humans, and that rather than constructing human-like generalizations from the training data, the system has merely memorized exactly what it has seen, based on some spurious characteristics. Although adversarial examples are just that – artificially constructed by "adversaries" and thus unlikely to occur spontaneously – their existence raises the concern that classifications formed by machine learning systems may not be naturally robust. That is, if a classifier may be significantly misled by very minor intentionally designed input modifications, it is possible that naturally arising, unintentional input variations may also lead to vastly incorrect classifications. Considered from this standpoint, the source of data variation – i.e., whether due to intentional adversarial attacks or normal input variations – is largely irrelevant when considering robustness. A system that is highly sensitive to input variability will not be safe regardless whether that variability is caused by an intentional attack or by natural changes in system's environment. In other words, robustness may be considered as a system's characteristic that is independent from the adversarial or non-adversarial nature of data variability.

In the sections below, we introduce concepts from robust and adversarial machine learning that should be considered when adapting design assurance and certification processes to include non-deterministic machine learning systems. These are considered for both a classifier and a dynamical system example. As Benjamin Recht stated, "Understanding how to properly analyze, predict, and certify such systems requires insights from current machine learning practice and from the applied mathematics of

optimization, statistics, and control theory."[38] The very nature of machine learning algorithms makes it difficult to predict and analyze their behavior, so we cannot build a fully principled defense against adversarial attacks (or, dually, verification of their absence), However emerging research in this field is leading to improvements in demonstrable system robustness that will help assure the safety of future systems.

3.2.1 Definition of Robustness

What is understood as robustness depends largely on the system's intended function. The two main cases that may be considered are discrete-output classifiers, and continuous-output regression algorithms.

In case of regressors, robustness is synonymous with bounded input-output sensitivity. One possible definition of a regressor that is robust with respect to input variations is that the following condition must hold for some properly selected δ and ε:

$$|f(x+\Delta) - f(x)| \leq \delta \quad \text{if} \quad \|\Delta\| \leq \varepsilon$$

It is readily seen that this notion of robustness is closely related to continuity of the regression mapping f. A regression function that is not continuous will naturally be not robust, as infinitesimally small input variations may lead to finite step changes in output, resulting in infinite input-output gain. Of course, the converse does not follow: a system that is continuous need not be necessarily seen as robust. The exact meaning of the robustness condition depends on how the allowed output variation δ is related to the expected input variation ε. In one case, a linear relationship $\delta = K\varepsilon$ may be specified, leading to the following robustness condition:

$$|f(x+\Delta) - f(x)| \leq K \, \|\Delta\|$$

This is equivalent to the regressor function being Lipschitz continuous, and the constant K may be interpreted as the worst-case input-output gain of the system. This highlights the connection between robustness and bounded gain; roughly speaking a robust system has a low input-output gain. In this view, assuring robustness is equivalent to limiting the gradient of the trained mapping f. A number of well-known regularization techniques may be applied to assure that the gradient remains bounded and below a prescribed value. In this sense, training a robust regressor is relatively trivial.

It is worth noting that increasing the system's robustness via lowering its input-output gain does not necessarily have to be beneficial or desirable. An absolutely robust regressor will have zero input-output sensitivity and will not be reacting to any input variations at all. In any practical application, the very purpose of employing a machine learning regressor is to produce different outputs for different inputs. A trained regressor should have its input-output gain sufficiently high to represent the desired system behavior, but low enough to preclude undesired behavior. The appropriate specification of the desired δ, ε relationship depends therefore on *a priori* knowledge of the system's environment and its performance requirements.

In the classification case, the common robustness definition requires that for a particular input x and an appropriately specified ε the following condition holds

$$f(x+\Delta) = f(x) \text{ if } \|\Delta\| \leq \varepsilon$$

[38] Recht, B., "A Tour of Reinforcement Learning: The View from Continuous Control." CoRR abs/1806.09460 (2018): n. 2. https://arxiv.org/pdf/1806.09460.pdf, retrieved March 1, 2019.

Intuitively speaking, a classifier is robust near an example x if its output remains constant within an ε neighborhood of x. Note that this concept of robustness is closely tied to a prescribed set of input examples of interest. A classifier can be robust at a finite number of examples but *cannot* possibly be robust *everywhere* within its input domain. The only classifier that is robust everywhere would have to be outputting the same value for all possible inputs, i.e. would not be able to classify at all. For a classifier to be able to classify, there must be at least one classification boundary on which the output value changes from one class to another. On such boundaries, the classifier in question will obviously be not robust.

The definition of robustness often depends on the choice of the norm used to characterize the size of the input variation. There are many different formulations for a norm that may be suitable for different applications of machine learning. For image processing, the L_∞ norm is often found most useful, but the ultimate choice of a norm is dependent on the application.

For the above reason, training a robust classifier depends on the particular training (or validation) set used. A classifier that is certifiably robust on one training set may turn out to be not robust on even slightly different training set, even if both sets are constructed with the very same application in mind. This dependence of robustness on the training set is in stark contrast to the regressor case. It is quite possible (and indeed desirable) to have a regressor that is uniformly robust in its entire input domain. On the other hand, a uniformly robust classifier is impossible by definition; it can be robust only with respect to a particular training set on a collection of disjoint neighborhoods.

3.2.2 Training Considerations for Improving Robustness

As explained above, training a robust regressor is conceptually trivial and equivalent to simply limiting the input-output gain, or the slope of the response surface on the trained algorithm. On the other hand, training a robust classifier is both non-trivial and inextricably tied to the selection of the training set. Accordingly, the latter problem has received significant attention and there are some interesting techniques available to assure robustness for specific cases. Here we review a few examples and discuss their implications for the general problem.

Wong and Kolter[39] address the case of a multi-layer ReLU network, using the robustness definition discussed in the previous section, with the input perturbations Δ restricted to "adversarial polytopes" $\|\Delta\|_\infty \leq \varepsilon$. For a finite number of examples with a finite number of target classes, their technique aims to assure that classifier's output remains constant within an ε polytope surrounding each example x. As such, a trained network is robust only on a finite set of ε neighborhoods tied to a particular training set.

The training method of Wong and Kolter makes a clever use of the piece-wise linear nature of a ReLU network. Output of each node is bounded by three linear constraints which allows using linear programming to find bounds on network output, given the assumed bounds on its inputs. A robust training algorithm is proposed that minimizes the upper bound on the loss function over all adversarial polytopes. The most attractive part of the method is its approach to verify whether the network is robust in the vicinity of a particular training example – readily addressable by the use of linear programming. Those examples that are identified as robust are said to have been issued robustness "certificates" – i.e. they are guaranteed to maintain the same value of the classifier's output within the entire ε neighborhood. Interestingly, the training method does not guarantee that all training data points will receive such certificates. For some examples there will be no robustness guarantees, depending on how the training algorithm converges.

[39] Wong, E. and Kolter, J. Z., "Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope", June 8, 2018. https://arxiv.org/abs/1711.00851, retrieved March 1, 2019.

It is worth noting that the method of Wong and Kolter assures robustness with respect to any input perturbations that are smaller than ε, regardless of the adversarial or non-adversarial nature of such perturbations (this despite the "adversarial polytope" title). This underscores our previous remarks that guaranteed robustness is independent from the concept of adversarial attacks. The benefits of this technique are significant, as adversarial attacks can be conducted with very imperceptible perturbations and mislead the classifier in most cases. An example in Figure 10 shows how a perturbation with a very low norm value may change the classification from correct ("butterfly") to incorrect ("hook", "claw").[40]

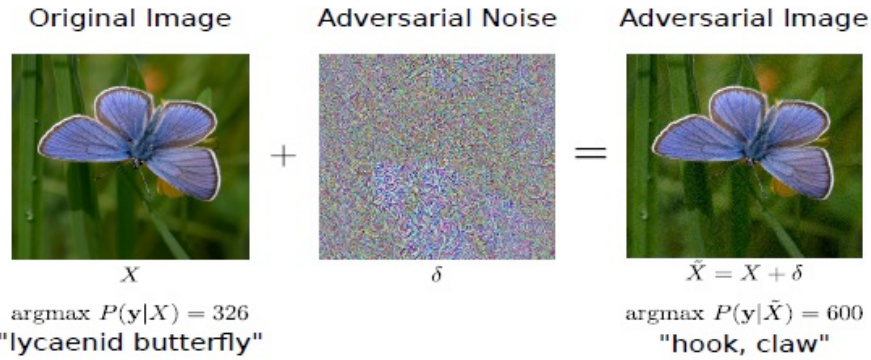

Figure 10: Example of an adversarial attack.

It should be noted that the clever use of the piece-wise linear nature of a ReLU network in this method is essentially a rebranded and revamped reincarnation of the hinged-hyperplane regression method that was well known in mid-1990s.[41] This underscores a general observation that guaranteed assurance of robustness is far easier for specialized forms of machine learning whose architecture features allow simplification of analysis. It remains to be determined what real-world applications allow such simplifications.

Another example of a robustness-focused training algorithm is given by Raghunathan, Steinhardt and Liang,[42] who use the same definition of classifier robustness based on $\|\Delta\|_\infty \leq \varepsilon$. Their method is not limited to piece-wise linear nodes and in fact allows arbitrary differentiable (or even almost-everywhere differentiable) activation functions. However, the method is limited to architectures with single hidden layers only. The method approximates an upper bound of network's growth within each ε neighborhood and uses a penalty term within the loss function to limit that growth term.

Just like in Wong and Kolter, the method of Raghunathan, Steinhardt and Liang does not guarantee robustness around all training examples. If the network is found to be robust around a particular example, that example is given a certificate. However, for some examples there may be no robustness guarantees.

Again, despite referring to "adversarial examples," the method is applicable to *any* input perturbations bounded by ε. There is nothing particularly "adversarial" about the technique and the name seems to

---

40 Hafemann, L.G. and Nguyen-Meidine, T., "Adversarial examples in Deep Neural Networks". https://www.etsmtl.ca/Unites-de-recherche/LIVIA/Seminars/lgh_adversarial_examples.pdf, retrieved May 26, 2019.

41 Ripley, B. D., Pattern Recognition and Neural Networks, Cambridge University Press, 1996. http://www.stats.ox.ac.uk/~ripley/PRbook/.

42 Raghunathan, A., Steinhardt J., and Liang, P., "Certified Defenses Against Adversarial Examples," January 29, 2018, https://arxiv.org/abs/1801.09344. Retrieved May 18, 2019.

be used solely to fit the paper into the popular field. Instead, the method has wider applicability to input variation problems that may include sensor noise or other random perturbations.

It is also worth noting that this method too depends on a special network structure – namely the single hidden layer. While mathematically a single hidden layer is always sufficient to approximate any regression or classification problem to arbitrary accuracy,[43] the current practice is to take advantage of multilayer architectures to leverage the available modern training algorithms. The method of Raghunathan, Steinhardt and Liang does not allow such architectures. This again underscores the apparent tradeoff between the network architecture simplicity and the ease of assuring robustness.

Another common feature of the two robustness-focused training algorithms is their dependence on the choice of the neighborhood size ε. Neither paper offers useful guidance on how to choose ε. This is not surprising, as this choice will depend on the particular application domain; in some cases, it may not be advantageous to have ε that is too large. This points to a general observation that all robustness analysis must be done in context of a particular application and its requirements.

3.2.3 Testing of Robustness in Deep Learning Systems

As discussed above, there have been promising advances in training for limited robustness for specialized network architectures. For the widely popular multi-layer deep learning case, the literature focuses on testing to ascertain robustness. Below we review a few relevant examples.

Pei, et al.[44] define the neuron coverage criterion as the ratio N_a / N, where N_a is the number of neurons activated by a test set, and N is the number of neurons in the entire network. The testing process is then defined as an optimization problem that maximizes neuron coverage by leveraging the fact that neural networks are trained using gradient descent, and thus the search for test inputs that yield different paths through a program may be guided by gradient ascent since all weights are known after training.

Tian, et al.[45] extended the neuron coverage testing method to also include convolutional layers and recurrent networks. The paper raises concern that inputs generated in Pei may not be realistic enough, and thus focuses on test generation techniques that produce more realistic inputs. They focus on real-world input variability phenomena related to autonomous driving such as camera lens distortion, weather conditions, and object movement. They also acknowledge that many applications of deep learning lack detailed specifications, which is an essential ingredient in verification activities. To avoid this issue, they leverage metamorphic relations between different synthetic images. For example, the steering angle of a self-driving car shall not be (significantly) affected by changes in weather conditions. The paper reports some examples where their Deep Test method discovered erroneous behavior in state-of-the-art neural networks trained for autonomous driving.

The issue of coverage criteria is further addressed by Li, et al.[46] The paper asserts that from the design point-of-view there are fundamental differences between deep neural networks trained using machine

---

[43] Hornik, K., "Approximation Capabilities of Multilayer Feedforward Networks", Neural Networks, 4(2), 251–257, 1991. http://zmjones.com/static/statistical-learning/hornik-nn-1991.pdf, retrieved May 15, 2019.

[44] Pei, et al., "Deepxplore: Automated Whitebox Testing of Deep Learning Systems." Proceedings of the 26th Symposium on Operating Systems Principles. ACM, 2017. https://arxiv.org/abs/1705.06640, retrieved May 15, 2019.

[45] Tian, Y., et al., "Deeptest: Automated Testing of Deep-Neural-Network-Driven Autonomous Cars." Proceedings of the 40th International Conference on Software Engineering. ACM, 2018. https://arxiv.org/pdf/1708.08559.pdf, retrieved Feb 20, 2019.

[46] Li, Z., et al., “Structural Coverage Criteria for Neural Networks Could Be Misleading,” ICSE 2019 New Ideas and Emerging Results, Montreal, CA, May 31, 2019,

learning, and software written by humans. Therefore, detecting erroneous behavior in neural networks is different from detecting those in traditional software, which necessitates novel test generation approaches. Still, the robustness determination should also investigate the consequences of "dead-neurons areas" or otherwise inactive decision gates in a machine learning system. Observability of hidden nodes, covered in Section 3.3, also may have impact on robustness. More detailed understanding of why some nodes are activated and others are not may provide a means to improve the robustness of machine learning systems

These three papers illustrate shortcomings of testing approaches available in literature. Each study defines its own coverage criteria and provides some comparisons to others. However, research on hierarchies of these criteria is missing, and connection to previously existing criteria such as decision coverage is insufficient. Li, et al. claim that existing coverage criteria such as branch coverage completely lose effectiveness when applied to deep neural networks, however without any evidence to substantiate that claim. Pei et al. compare neuron coverage with code coverage of a neural network execution engine while running tests cases. This type of code coverage is not sufficient, as it is analogous to measuring the coverage of a hypervisor to assess the quality of each isolated operating system.

The papers reviewed above provide strategies to generate test inputs and demonstrate that it is easy to come up with examples that violate the intended behavior of neural networks. The lack of detailed specification (and thus a testing oracle) still seems to be an open research topic.

3.2.4 Summary Observations and Research Needs

The concept of machine learning robustness is tightly linked to the notion of continuity and boundedness of gain, for regression networks, and with local lack of sensitivity, for classifiers. In the latter case there is a fundamental tradeoff between local robustness and the need for accurate discrimination between different classes: the more robust a network is, the less sensitive it will be with respect to discriminating features. The most popular approach to define and analyze robustness is via specifying the size ε of adversarial neighborhoods. The larger those neighborhoods, the less responsive the network will be. Thus, training for robustness requires intimate knowledge and understanding of the application domain requirements.

There is a tradeoff apparent in the available literature between the complexity of network architecture and the available training apparatus to improve (not even assure) robustness. For piece-wise linear (ReLU) and single hidden layer architectures, attractive robustness-focused algorithms have been proposed. For the more popular and more versatile multi-layer deep learning architectures, the literature focuses merely on verifying or testing for robustness. There seems to be a need for robustness-focused deep learning methods.

Robustness is also tightly linked with the detection (or non-detection) of characteristics or properties in a classifier that are not easily quantifiable, but are crucial for classification. For example, how many wheels or legs can be "seen"? This automated kind of data processing is something that humans perform without even noticing, which provides robustness to adversarial attacks as illustrated in the images above. Such properties would be instantly recognized by a mammal brains but are still challenging to mimic within machine learning. This phenomenon reflects the existing difficulty of tracing high level requirements to a more granular understanding of the operation of machine learning systems. Improvements in this area will help with the introduction of machine learning elements in certifiable aerospace systems.

---

http://202.119.32.195/cache/11/03/moon.nju.edu.cn/c10bc72b8639f971f414a34620e26bca/2019-ICSENIER-ZLi-Misleading.pdf, retrieved July 31, 2019.

Currently available techniques are academic and scalability to real-world problems remains an issue. Rapid adoption of machine learning technologies in aerospace systems will require detailed considerations of the robustness of these systems, and more importantly the impact of the lack of verifiable robustness on the safety of the system in all foreseeable operating conditions. Such systems may operate well in advisory functions, but additional research is required before robustness can be assured for safety-critical systems.

3.3 Observability and Interpretability

While machine learned models have shown impressive results in a wide range of problem domains from image classification, object identification and tracking, motion planning and control such as those present in self-driving cars, and natural language processing, most techniques are inherently black box in nature. While access to the internal structure of a machine learned model (i.e. the weights of a neural network, for example) is readily available, it can be difficult to interpret the emergent logic of such systems. These limitations and non-intuitive traceability (particularly from the perspective of current traceability practices) makes certification of data-based ML within safety-critical systems an open challenge.

The following subsection and associated appendix provide an overview and discussion into the current state of AI Interpretability. Specifically, this section aims to provide rationale for the importance of such techniques and clarity regarding terminology. Additionally, a high-level technical explanation and relevant use cases are provided within the Appendix. The intent is to shed light on potential applications of AI Interpretability within aviation, based on current usage of such techniques in similar domains like the automotive industry.

3.3.1 Relevance of Interpretability Techniques

There are a number of reasons for increased attention, research and incorporation of interpretable AI techniques. While research into interpretable AI is nascent, initial insights generated provide promising results for the future. Below is a list of potential benefits from developing interpretable systems:

- **Support for evidence-based certification:** Such techniques could provide support for evidence-based certification via human-like communication of competency or generating rationale for decisions. Interpretable AI techniques are already being utilized by the automotive industry during development to provide visual evidence of navigation decision making in self-driving cars.[47] More research, development and assurance on interpretability is still needed, particularly as methods for enabling machine learning and especially deep learning models to be more interpretable are unsatisfying for those accustomed to hard constraints and provable bounds on system operation. The expanding collection of methods for explainable insights are primarily heuristic, enabling the developer to test hypotheses about the model's performance, but not to extract hard rules and closed form expressions.
- **Aid in human machine teaming:** through more robust communication between human and artificial intelligence. While full automation is an exciting goal within some industry tasks, it is likely that AI will also be highly prevalent as a mechanism for assisting human decision making and control. In such cases, it will be increasingly important for AI to appropriately communicate and transition control bi-directionally with a human as appropriate.

[47] Bojarski, M. et al., "Explaining How a Deep Neural Network Trained with End-to-End Learning Steers a Car," Apr. 25, 2017, paper given at ICLR 2018, Vancouver, CA. http://arxiv.org/pdf/1704.07911v1, retrieved May 16, 2019.

- **Engendering Trust:** Engendering and maintain trust within users requires enhanced communication (visual, textual, vocal, etc.) and by recognizing and addressing mistakes in a timely fashion. By way of example, a previous study looked at the effects of trust on humans with a robot guide providing assistance to the nearest emergency exits. Through the study, it was shown that as intuited, when a robot made navigation errors human trust was degraded and usage was reduced.[48] Yet, when the robot appropriately explained the reason it failed or apologized for the failure with a properly timed response, the subject's trust was repaired.[49]
- **Avoiding latent biases** and unintended side effects on predictions through inadequate training data. Accuracy of an underlying ML model does not guarantee appropriate predictions. Specifically, a high prediction rate may result from quality and bias issues of the training data. In the military trained a classifier to recognize enemy tanks from friendly tanks.[50] The classifier resulted in a high accuracy on the test set, but poor performance when deployed in the field. It was later discovered the ML model utilized unintended features (non-tank related) in the dataset to classify images. Enemy photos were taken on overcast days, while friendly photos on sunny days.
- **Enhancing effectiveness** of black box techniques by pointing to evidence for specific predictions/outcomes. By exposing the black box systems, researchers can start to spot pitfalls within model's internal logic earlier in the development process, and subsequently construct strategies that address the undesirable.

AI Interpretability will certainly become an increasing requirement preceding deployment of machine learned systems. The European Parliament recently adopted the General Data Protection Regulation (GDPR), which became law in May 2018.[51] "An innovative aspect of the GDPR…are the clauses on automated (algorithmic) individual decision-making, including profiling, which for the first time introduce…a right of explanation for all individuals to obtain 'meaningful explanations of the logic involved' when automated decision making takes place."

3.3.2 Use Cases for Aviation

As an example, AI interpretable techniques could be utilized by both end users and system designers within the aviation industry. Such techniques could one day be used to interpret model decisions and provide explanation during autonomous taxiing, or provide textual or vocal responses during off nominal situations within the cockpit, while simultaneously pointing to evidence/data as justification for recommended course of actions. Additionally, interpretable AI techniques could help algorithm developers understand and identify the causal features generating diagnosis related to predictive maintenance applications.

---

48 Robinette, P., Howard, A., and Wagner, A. "Timing is Key for Robot Trust Repair," Proceedings for 7th International Conference on Social Robotics, ICSR 2015, Paris, France, Oct, 25-30, 2015, pp 574-583, Springer. https://sites.psu.edu/real/files/2016/08/Trust_Repair_ICSR-Submitted-1sl4k5h.pdf and https://www.sciencedirect.com/science/article/pii/S1367578814000406, retrieved on July 31, 2019.

49 Lyons, J., et al., "Certifiable Trust in Autonomous Systems: Making the Intractable Tangible", Artificial Intelligence Magazine, 38, (3), pp. 37-49, https://www.aaai.org/ojs/index.php/aimagazine/issue/view/219, retrieved May 21, 2019.

50 Branwen, G., "The Neural Net Tank Urban Legend," https://www.gwern.net/Tanks, retrieved July 31, 2019.

51 "2018 Reform of EU Data Protection Rules." European Commission - European Commission, May 22, 2019, https://ec.europa.eu/commission/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules_en, retrieved June 21, 2019.

3.4 Research Summary - Safety Systems Using Machine Learning

During the course of this project, two types of literature searches were conducted: 1) an ad hoc search conducted by the members of the PMC and the sub teams formed during the course of the project, and 2) a formal search conducted using defined search criteria and the reference library resources of the Texas A&M University. Both searches revealed extensive literature in the general area of machine learning algorithms and methods. There are relatively fewer papers on application of machine learning in safety-critical applications, however, new papers are emerging as especially the automobile and aerospace industries consider adoption of machine learning methodologies.

The project team reviewed what similar industrial domains are doing with "certification" of these cyber-physical systems, citing the automotive industry, the medical industry, and the nuclear industry – all domains with strong interest in certification of safety-critical systems. There is a gap in coverage of verification and validation aspects of machine learning approaches and algorithms; there is an apparent dearth of research in these areas.

In conducting the literature search, the project team took a systematic three-step approach as shown in Figure 11.

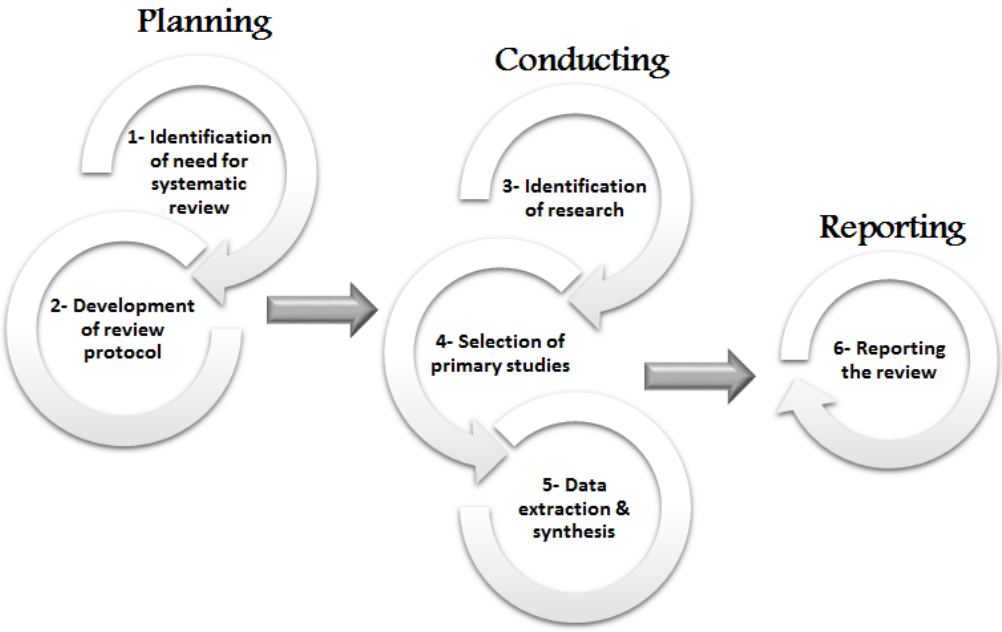

Figure 11: Literature search process.[52]

Both the ad hoc and formal searches conducted by this project team focused on those papers applicable to safety-critical applications. Over 150 papers were reviewed for relevance by team members. A full listing of the references is provided in a separate indexed spreadsheet that can be obtained from the AVSI project archives. Contact AVSI at https://avsi.aero.

[52] Galavotti I. (2019) "Systematic Literature Review on Experience and Learning in Acquisitions: Search Strategy and Data Synthesis." Experience and Learning in Corporate Acquisitions. Palgrave Macmillan, Cham.

4 Machine Learning Process Flow for Safety-Critical Applications

4.1 Introduction

This section of the document describes what project members expect would be a representative life cycle for development of an ML-based system. It is intended to serve as a starting point for further discussion, detailing or change. The process flow also suggests minimum criteria for some of the proposed activities.

4.1.1 ML Development Life Cycle Process

The Machine Learning Life Cycle process may be described as consisting of the following activities:

1. Define a set of requirements that describe what is to be implemented, the desired behaviors, performance and acceptable margins of error.
2. Develop and manage with rigorous configuration control the datasets to be used to train the ML subsystem, based on the data types defined by system input requirements. These datasets will be broken into three separate categories:
    a. **training** – the chosen machine learning topologies must be trained to establish parametric values for all nodes and connections,
    b. **test** – used to check if the trained topologies behave as expected with data previously not seen, and
    c. **verification** – used to verify the behavior of the executable code once it has been integrated into the target platform/system.
3. Provide a set of possible paradigms [Support Vector Machines (SVM), Neural Networks, etc.] eligible to solve the problem at hand, as presented by the set of requirements.
4. Select from the possible paradigms, the most applicable candidate for the problem at hand.
5. Select from the chosen paradigm the topologies that represent a solution that complies with the requirements. Training is performed by injecting training data on selected topologies. Modify/Update topologies as needed to comply with the set of requirements, based on the training data.
6. Requirements, such as accuracy, error margin and performance, should be used to select the best topology. Prune the ML algorithm to remove unnecessary nodes.
7. Inject test data on the trained model to guarantee that the design still complies with the set of requirements. This step is intended to validate that the selected topology still behaves as expected once exposed to new data.
8. Generate source and executable code from the trained model and integrate it into the target hardware.
9. Inject validation data in the target hardware to measure the level of compliance of the executable code with the data that represents a set of requirements.

Figure 12 illustrates the relationship between these expected activities.

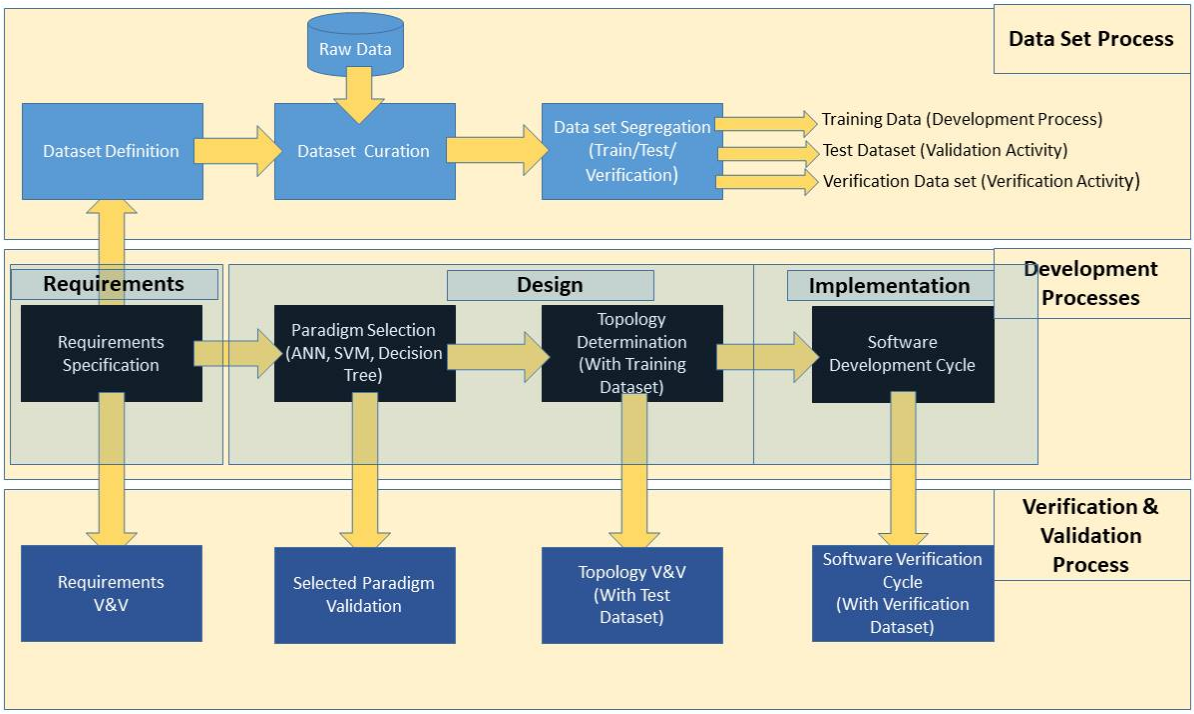

Figure 12: Machine learning development flow.

4.1.2 Requirements

Requirements may be developed as they are for any other given technology. They may be represented as text, as mathematical or logic expressions or as a model, for instance. However, it should be noted that ML-based systems might also require that some unique criteria be specified. Suggested unique criteria are listed below.

- **Behavior -** Concerns the operational envelope of the machine learning system and its robustness to foreseen and unforeseen inputs. The possible actions a machine learning system can perform, the constraints on those actions, and the internal and external data affecting system outputs and performance, need to be understood.
- **Training Process -** Concerns the software pipeline and methodologies for optimizing the machine learning system. The training process needs to consider the operational environment, type of data, quality of data, structure of data, machine learning model, the optimization methods, and metrics used to measure success of training. Training of a machine learning system is typically a continual process. Therefore, it is important to determine how models will be further optimized by new data, as well as understand how training on the new data may affect prior performance and robustness. Further, the 'stop criteria' for training needs to be considered. Stop criteria concludes the training process as appropriate for the given application in order to avoid over-fitting. It may consider performance thresholds, degradation of performance and runtime behaviors.
- **Performance Metrics -** As machine learning models are optimized against their respective loss functions, the performance of a model is expected to improve. Machine learning models are typically benchmarked via accuracy (supervised learning) and accrued reward or performance against operational goals (reinforcement learning), to name a few. Performance metrics need to

be accessed throughout the lifecycle of the system from training to deployment, particularly as performance can vary between simulated and/or training data to the real world environment.

- **System Inputs -** Concerns the data provided as input into the machine learning model ultimately used to make a prediction or perform an action. Considerations for the data utilized as inputs include: data format/type, methods for accruing/capturing data, the intended operational environment, differences in data between the training versus operational environment, and process for separating training, validation and test datasets. System inputs can be further used to inform safeguards against adversarial modifications to the data.
- **System Outputs -** Concerns the actions or predictions provided by a machine learning model based on the inputs received. The output interface needs to be well defined such that the range of outputs, format of outputs, and acceptable rate of errors is understood.

4.2 Process Flow – Machine Learning as a Design Aid

In certain applications, such as control loop optimization and tuning, machine learning methods may be used as a design aid during development. When machine learning is used only as a design aid, the machine learning algorithms provide the fixed design parameters used in a conventional implementation. Since the machine learning algorithms are not part of the runtime end system, the safety concerns are minimized: the runtime implementation can be verified and validated using accepted verification, and validation processes to assure the end system meets its safety objectives.

For example, tuning of the feedback loops used in complex flight control to achieve optimal performance can be a time-consuming design activity using a combination of simulation, lab, and flight test results to balance all of the feedback parameters to optimally meet the performance requirements. Re-applying that control system to a modified or new platform often means repeating that same time-consuming optimization process. In some cases, it is expected machine learning may be a useful and efficient means to optimize parameters. Once the machine learning has determined the optimized parameters, those parameters can be fixed in the implementation and verified and validated according to conventional methods (for example, gain and phase margins, impulse response, etc.).

4.3 Process Flow – Fixed Runtime Parameters

This is expected to be the most frequently used process flow for initial application of machine learning functions in safety-critical systems. It is similar to the development life cycle and system deployment of traditional non-ML systems in that a design is “fixed” once the system is deployed in a certified application. For a system with embedded machine learning, this means all the training is completed during the design phase. Runtime learning is not permitted once a system is certified for use (see the next section for a discussion of runtime learning processes).

As an example, consider a safety-critical system that uses an artificial neural net in its implementation. The ANN would be designed, trained, tested, and implemented as described in 4.1.1. The ANN topology, weights, and biases would then be fixed and under configuration control. As shown in Figure 13, the implementation will be verified, integrated, and validated and then deployed to the field with those fixed parameters. Runtime changes to the ANN topology and parameters must be disabled to prevent changes and loss of configuration control.

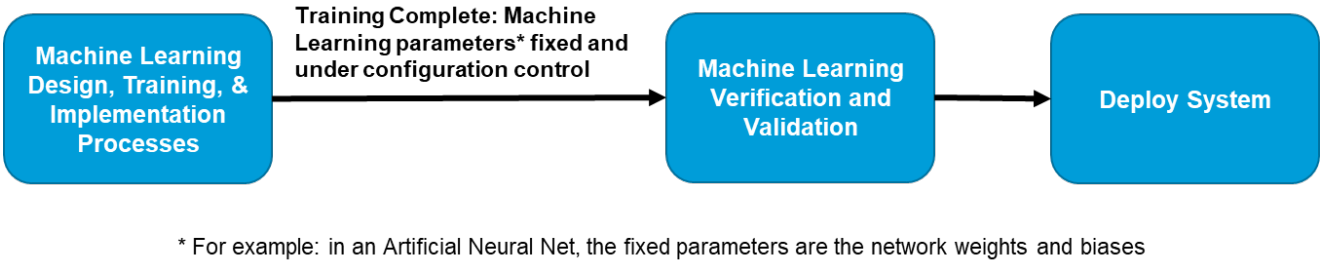

Figure 13: Process flow: runtime fixed parameters.

4.4 Process Flow – In Service Updates

Machine learning algorithms can have the ability to learn from runtime experiences, modifying future behavior on the basis of the continued exposure to new situations and ongoing assessment of past performance. In some applications, this behavior is desired to achieve continually improving performance while in service.

In safety-critical applications, however, any post-certification runtime modifications present challenges that must be addressed to assure continued air worthiness. These key issues include:

- **Loss of configuration control:** safety-critical systems must be maintained under configuration control.
- **Emergent system behavior:** any new emergent behavior must be assessed for its impact on continued safe operation.
- **Consistency of operation:** consistent system behavior is important for training and standard operating procedures.
- **Unintended behavior:** there must be guarantees that post-deployment learning cannot lead to unintended behavior or “mis-learning”.

Until more field experience is gained with “fixed” machine learning processes, it is recommended that in-service modifications be re-certified prior to being deployed. This still allows in-service data to be used, but in a controlled fashion that ensures the safety objectives continue to be met with the updated system.

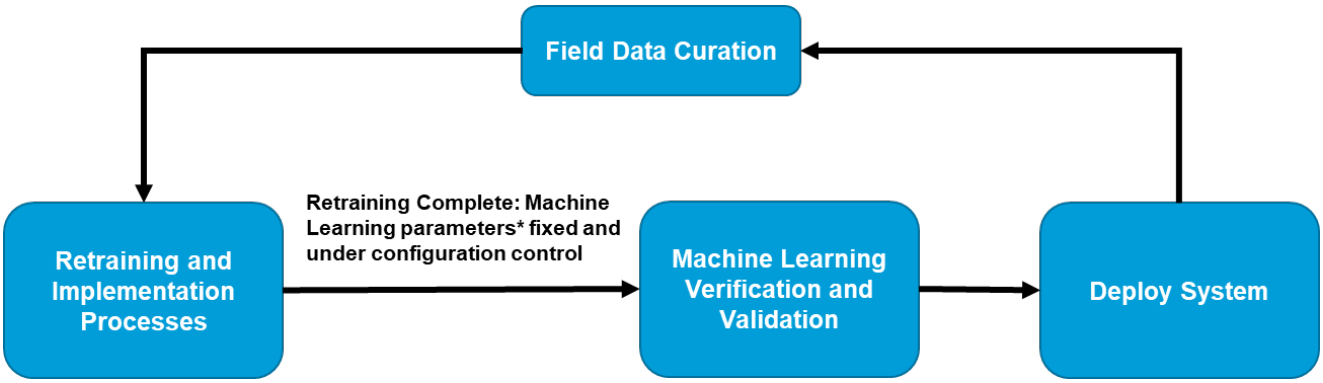

Figure 14: Process flow: in service updates.

The recommended process is shown in Figure 14. Field data is collected and curated from the initially deployed system. The curated data is used to cycle back through the training and implementation processes. Once the retraining and implementation updates are complete, the “fixed” system is re-verified, re-validated, and then re-deployed to the field.

4.5 Machine Learning Datasets Considerations

Datasets are the primary means for training and verifying a machine learning topology. Datasets may be created or may be based on recorded raw data. As discussed above, the dataset (Figure 12) is a key strand for the training, testing and verification steps for machine learning models. Dataset quality has a direct impact on the performance of the ML-based system. A poor quality or poorly managed dataset has the potential to make a machine learning model perform poorly and output behavior to vary unacceptably.

Section 4.1 emphasizes the aspects of a curated dataset and its characteristics. However, in terms of the machine learning development process, datasets are closely related to system requirements in the sense that requirements capture expected input types, ranges, formats and tolerances for each type of input. Additionally, depending on the ANN being developed, datasets may include expected outputs for specific input data vectors. Expected errors should also be defined in requirements, since they may be used for robustness validation of the chosen topology.

Datasets should be carefully verified in order to guarantee that they represent the whole range of expected operations. Data that is representative of only a subset of the foreseeable range of operations may lead to bias in the response of the selected topology, and therefore may lead to unexpected responses when the topology is subjected to inputs that are outside the range of data considered during development.

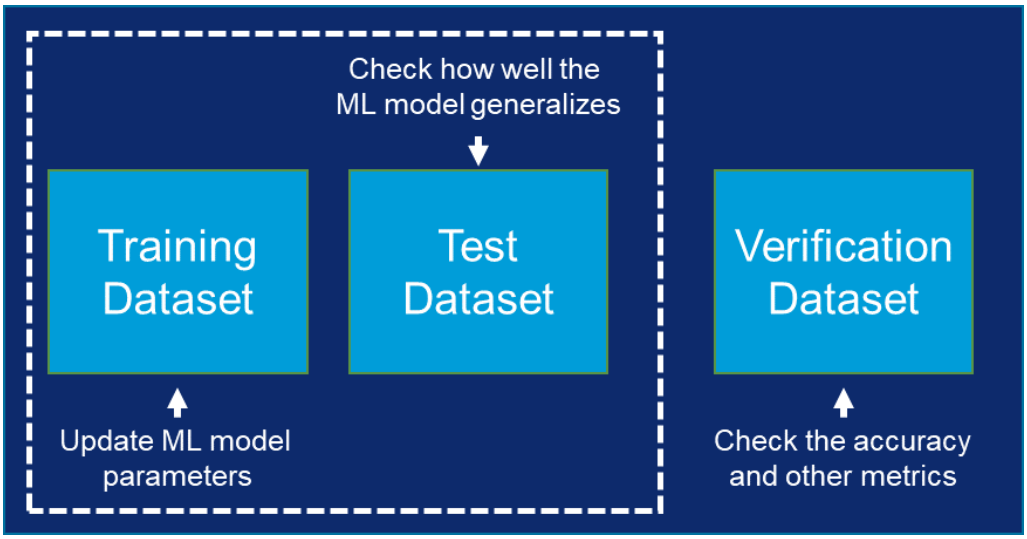

Figure 15: Dataset segregation for machine learning models.

Datasets which are based on raw recorded data may be prone to noise or missing information. It is up to the development team to decide on how this data is to be handled. For example, interpolation, mean values, or constant values may be used to fill gaps in data. The team might even decide to ignore certain recorded parameters entirely. It should also be noted that datasets may be refreshed through the data curation and configuration management process with new data in order to update this baseline data for training, validation and test purposes.

As illustrated in Figure 15, dataset segregation is used for three different purposes:

- **Training:** used during the design process as a means to determine the weights and biases within a given topology that produce correct outputs for a given set of requirements.
- **Test:** used during the design validation process, by submitting the chosen topology to new data and checking if it still behaves as expected according to requirements.
- **Verification:** to verify that the software complies with its set of requirements after integration with the target hardware.

Based on the discussion above, datasets may be subject to a minimum set of quality criteria in order to be considered sufficient for development of ML-based safety-critical systems. The criteria listed in Table 2 provide a potential basis for future standardized safety objectives useful for guiding development of systems built from ML datasets.

Table 2: ML training/testing & verification data criteria.

Data should be representative of the problem under analysis.
Training and Verification Data should be representative of the domain usage.
Data inputs should be verified against expected format.
Data inputs should be verified for discrepancies in values or missing data.
Data inputs should be checked for dependencies on other inputs.

4.6 Design

The design phase is composed of two sequential sub-phases: Selection of Applicable Paradigms and Selection of Applicable Topology.

- **Applicable Paradigms:** Machine Learning encompasses multiple paradigms, each of which have particular strengths and weaknesses. The first step in the design phase is to determine which technique will best satisfy the set of requirements associated with the function being implement with the ML-based system. Selection criteria may need to be established and a rationale for the decision should be recorded for future reference. It is likely that selection of a specific paradigm will lead to a second level of derived requirements, since the choice of paradigm determines additional design details that need to be specified.
- **Topology:** is the detailed structure of the type of ML paradigm chosen to solve the problem as expressed by the initial set of requirements. A trained model is an *instantiation* of a topology. The trained model is the source for generated source and executable code that can be integrated and verified on the target platform/system. It is up to the development team to decide how many different topologies should be analyzed before defining the best fit. Criteria used in selecting a topology include training rate, error margin, performance and other requirements. Once a final candidate is chosen, it must be validated with the segregated validation dataset. For example, if the selected topology is a neural network, it should have its weights and biases frozen, as well as the number of inputs for each level and the number of outputs defined for the last level.

Designers must verify that the chosen paradigm and topology comply with the set of functional and performance requirements defined for the ML-based system.

4.7 Deployment

While it is possible that a ML development does not result in a deployed system (see Section 4.2 for example), the deployment phase typically refers to activities required to embed a ML-based system in a safety-critical application. These might include the generation of source code, executable code, hardware acceleration, and integration in the target platform/system. These activities employ processes that are covered by existing standards such as RTCA DO-178C and RTCA DO-254, and as such are not within the scope of this document.

4.8 Validation Process

The validation process is used to guarantee that for each phase of the development life cycle, the ML-based system is still performing it's intended function. Figure 16 illustrates the flow and relationship between validation activities and artifacts.

The following elements of the ML model are subject to validation:

1. **Requirements –** as defined in the usual sense and subject to validation activities such as those defined in SAE ARP4754A.
2. **Training/Test Data and Verification Data –** In this case, these separate datasets need to be validated in terms of data types, data inputs and their formats, comply with requirements, and if there are any missing or discrepant data. Data may also be validated in the sense that they should be representative samples of the whole operational envelope that the topology will face.
3. **Selected Paradigms –** Review of selected paradigms and the respective rationale for their selection.
4. **Selected Topology –** Topology may be subject to both normal and robustness test cases by running scripts that will inject test data. Comparison against expected results will validate the solution in terms of performance, functional behavior and error margins as defined in the set of requirements.
5. **Implemented Solution –** Verify implemented solution on target hardware against verification data. In this case, source code, executable code and integration may be subject to specific standards, such as RTCA DO-178C tables A-5, A6 and A7.

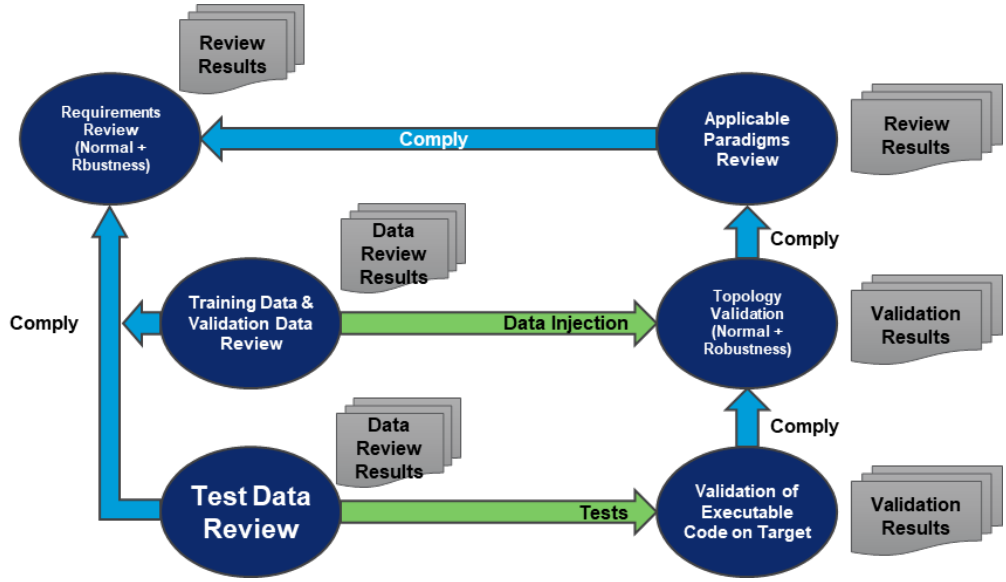

Figure 16: Machine learning validation.

4.8.1 Validation Artifacts

The following artifacts may be generated for the activities presented in Figure 16:

- The Validation and Verification of Requirements activity produces a baseline recorded as a Verification and Validation Report.
- The Review of Baselined Training and Test Data activity produces a Review Record & Traceability to Requirements Baseline Report.
- The Review of Paradigms and Respective Rationale activity produces a Review Record & Traceability to Requirements Report,
- The Review and Analysis of Baselined Topology activity produces a Review and Analysis Report & Traceability to Requirements and Paradigms Report,
- The Review of Baselined Verification Data activity produces a Review Record & Traceability to Requirements Baseline Report,
- The Review and Analysis of Verification performed on the baselined implemented solution produces a Verification Report and Traceability Report to the baseline set of requirements.
- Testing produces various test reports.

4.8.2 Configuration Control Process

Configuration Control should be exercised throughout the ML development life cycle. Requirements, datasets, applicable paradigms and selected topologies should be subject of configuration control as shown in Figure 20.

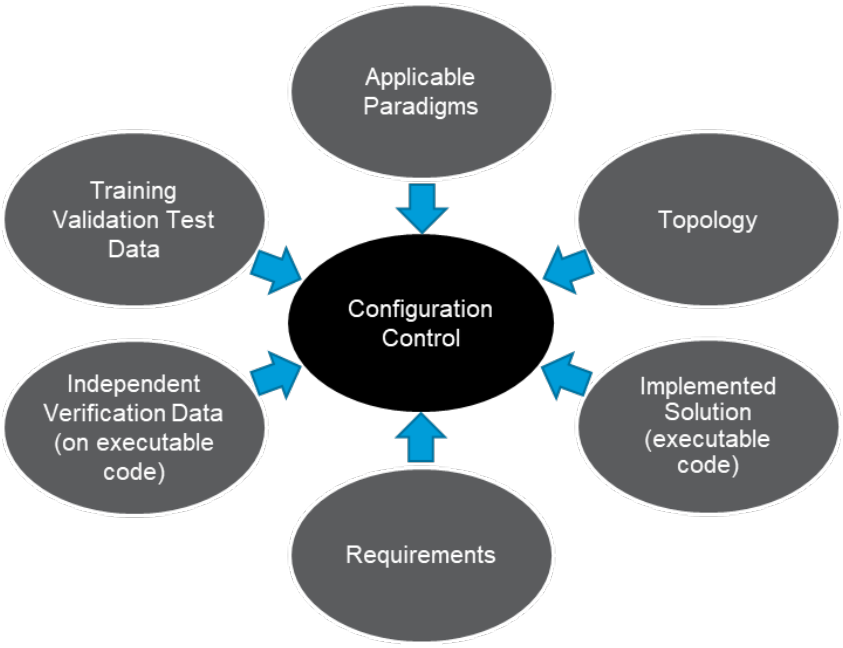

Figure 17: Configuration control.

4.8.3 Quality Assurance Process

Quality Assurance is an existing engineering process that applies equally to ML-based system development. As such, it should be actively applied throughout the entire development life cycle in order to guarantee that development processes are compliant with plans and adopted standards.

5 Perspectives on Assurance

This section presents some perspectives on design assurance relevant to the development of ML-based systems. It is intended to integrate some of the lessons learned during this project within a broader discussion of design assurance of complex systems. To set the framework for the discussion, a summary of current assurance practices is briefly summarized below.

5.1 Current Practice

The current aerospace development approach is largely guided by documents published by SAE International and RTCA, Inc. At the system level, SAE recommended practice ARP4754A – *Guidelines for Development of Civil Aircraft and Systems* covers the complete aircraft development cycle. The guidance places a strong emphasis on safety, requirements definition, validation, and architectural justification. ARP4754A classifies hazards into five levels A through E defined by the criticality of the hazard, with level A denoting catastrophic hazards, and level E denoting hazards with no safety impact. A complementary document, ARP4761 – *Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*, describes hazard and safety analysis processes where hazards are identified and assigned criticality levels via a functional analysis process, the result of which allocates a Function Development Assurance Level (FDAL) to each function. The FDAL provides the basis for the rigor of the development assurance processes in accordance with the criticality of the hazard. ARP4754A also recommends that architectural mitigation strategies be deployed to mitigate failure conditions which may contribute to hazards. Depending on the architectural mitigation and redundancy configuration, each of the hardware and software items that implement the architecture is an assigned an Item Development Assurance Level (IDAL). Similar to FDAL, IDAL designates five assurance levels A through E. The IDAL determines the rigor of the hardware and software item-level implementation development assurance processes. These processes are prescribed by guidance under RTCA in the form of DO-178C (for software items) and DO-254 (for hardware items).

The item guidance documents outline processes that assure the correct implementation of the respective hardware or software item. Hence, both processes assume validated requirements are developed through the system development process. The foundation of both the hardware and software item assurance processes is requirements driven design assurance. This mandates a high degree of design transparency and structural mapping to the driving requirements. Under DO-254, elemental analysis is used to demonstrate that each part of the design relates to the implementation of a requirement. Similarly, structural coverage analysis performed under DO-178C addresses the same objective for software designs. It is this combination of requirement driven verification with white box structural design transparency that provides the foundation for the dual claims of correct functional implementation and implementation of only the intended functions.

5.1.1 Safety Assessment and Assumptions Related to Software Reliability

As described above, ARP4761 supports ARP4754A objectives with specific recommended practice relative to the safety assessment processes. This document provides more detailed guidance on how to conduct the functional hazard analysis and preliminary and final safety assessments. The recommended practice draws from broadly accepted methods including Fault Tree Analysis (FTA), Failure Modes and Effects Analysis (FMEA), Common Cause Analysis (CCA), Zonal Safety Analysis (ZSA), and Common Mode Analysis (CMA).

Within the safety assessment processes, the assessment of the software contribution to safety is made on a purely qualitative basis. There is no quantitative accounting concerning software reliability. There

is, therefore, an implicit underlying assumption that the assigned IDAL, together with the respective development assurance processes that it drives, provide sufficient confidence to preclude a software design error from contributing to the functional hazards.

5.2 The Disruptive Nature of Machine Learning Implementations

Machine learning technologies disrupt the current assurance paradigm in two important ways. First, traditionally defined and allocated requirements are replaced by requirements represented by datasets. Secondly, traditional tools and processes used to analyze and develop a structural understanding of a system design are not well suited to the emergent behavior of ML systems.

The traditional system to item requirement flow down is replaced with requirements that are empirically encoded with training and datasets. This issue with ML technologies is disruptive to the current assurance paradigm in two ways. The first relates to the challenge of the specification itself. As Spanfelner points out, the functionality of ML/AI components may involve the perception of the environment, which is not entirely specifiable.[53] For example, what is the specification for recognizing a pedestrian or a car? A second challenge relates to the gap and sensitivity differences between human and machine interpretation of sensory datasets. This gap is illustrated by the recent work that shows that slight variations in the input data, not perceivable by humans, can lead to degraded/incorrect performance of the machine learning system.[54] Such issues may complicate and compromise human review, which is a cornerstone of the current item level design assurance processes, especially at higher DAL levels.

Therefore, at today's level of technology and industrial practice, a training set is not an adequate substitute for a specification. The training set may not be complete, and there is no guarantee that it is even representative of the space of possible inputs. While the data curation activities suggested earlier may go a long way helping to fill in this gap, more research is needed to establish a more formal line of evidence and justification. While one can infer the level of safety from a traditional system specification, the training set provides no guarantee about the (potentially infinite) set of input/output scenarios that the component can handle. Thus, a training set on its own may not be sufficient evidence for a safety argument.

The structural complexity (or obscurity) and the emergent nature of the ML implementation behavior no longer support complete structural understanding of the design. Hence, it is difficult to trace functionality to elements of the architecture and ascertain whether functionality is limited to that which was intended.

Research has shown progress in this area. Recent work applying formal methods to network verification shows promise. One such algorithm to verify the input/output behavior of a neural network presented by Katz et al. is Reluplex.[27] It is a sound approach to verify safety properties of deep neural networks with ReLU activation functions. Intuitively, the algorithm is a modification of the simplex algorithm for solving linear programs. It operates by solving the linear constraints posed by the neural network's weighted sums while attempting to satisfy the non-linear constraints posed by its activation functions. It has been successfully demonstrated using a family of deep neural networks designed to operate as controllers in the Airborne Collision Avoidance System (ACAS Xu). Further, Reluplex has been used to

---

53 Spanfelner, B., et al, 2012. "Challenges in Applying the ISO 26262 for Driver Assistance Systems," Tagung Fahrerassistenz, Munchen 15 (16): 2012, http://www.ftm.mw.tum.de/uploads/media/28_Spanfelner.pdf, retrieved June 21, 2019.

54 Pezzementi, Z., et al., "Putting Image Manipulations in Context: Robustness Testing for Safe Perception ," IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), Aug. 2018, https://users.ece.cmu.edu/~koopman/pubs/pezzementi18_perception robustness_testing.pdf, retrieved May 28, 2019.

evaluate techniques for finding and defending against adversarial learning.[55] However, the NP-hard nature of the underlying problem of verifying ReLU NNs makes Reluplex work only on small networks, such as the ACAS Xu implementation that has five inputs, five outputs, and 300 total hidden neurons.

However, given that this approach encodes the entire network, it is computationally infeasible for image-based NNs that have significantly more inputs, outputs, and complex architectures. For instance, even the simple MNIST example may have 630,016 ReLU nodes.[56] More recent advancements in this area are demonstrating some improvement. ReluVal was shown to perform up to 200 times faster than Reluplex and Neurify outperformed ReluVal by a factor of 20.[57,58] However, the applicability of these techniques on real systems has not yet been demonstrated. Nevertheless, as such techniques evolve, they may form the foundation for a new class of authoritative guidance that may guide machine learning certification.

5.3 Characterizing the Disruption

From the discussion above, it is evident that ML-based implementations remove the foundation of the current design assurance paradigm at the item level. A redistribution of confidence may best characterize the impact of this. For traditional, imperative software, existing item level assurance techniques have proven to be very effective at limiting design errors. Thus, the qualitative-only treatment of software within the system safety processes is arguably justified. However, at the current state of practice, techniques to argue an equivalent level of confidence within ML-based implementations are not available. Such applications may be considered to be black box software, and as argued by Butler et al.,[59] it is practically infeasible to achieve the level of confidence needed for safety-critical applications by test alone. Hence, additional research must occur to address this gap.

The reduction in confidence concerning guaranteed correct software behavior has additional impact above the item level. As discussed above, under ARP4754A architectural mitigations are justified within a context of the IDAL allocations to the implementation items.[1] Redundancy is principally justified as an acceptable means to mitigate hardware failures. Under such assumptions, software-based functionality can be replicated on top of multiple independent hardware components to mitigate common mode hardware failures for improved function availability and integrity. Once again, this was primarily supported by the underlying philosophy that the item assurance processes were “good enough.” This assumption may not hold if confidence in the correct software behavior is reduced. Hence, software replication and redundancy strategies may need to be adapted for ML-based systems.

Some of the architectural mitigation strategies discussed in this document may present some avenues for exploration, for example, the mix of machine learning implementations with traditionally implemented safety monitors. However, the analysis of the safe reversionary state or backup mode will

---

55 Gopinath, D., et al., “DeepSafe: A Data-driven Approach for Checking Adversarial Robustness in Neural Networks,” https://arxiv.org/abs/1710.00486, retrieved Mar. 18, 2019.

56 Huang, X., et al., “Safety Verification Of Deep Neural Networks”, International Conference on Computer Aided Verification, pp. 3-29, 2017, https://arxiv.org/abs/1610.06940v3.

57 Wang, S., et al., “Formal Security Analysis of Neural Networks using Symbolic Intervals.” arXiv:1804.10829v3 [cs.AI], Jul. 2018, https://arxiv.org/pdf/1804.10829, retrieved May 18, 2019.

58 Wang, S., et al., "Efficient Formal Safety Analysis of Neural Networks," Advances in Neural Information Processing Systems. 2018, https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-wang_0.pdf, retrieved May 24, 2019.

59 Butler, R., and Finelli, G., "The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software." IEEE Transactions on Software Engineering 19.1 (1993): 3-12, https://shemesh.larc.nasa.gov/paper-nonq/nonq-paper.pdf, retrieved May 16, 2019.

need meticulous attention. Any reduced or simplified functionality will need to be demonstrated to be sufficient for all anticipated operational scenarios. The cost impact of supporting adequate capability within the backup and monitoring system will also be a consideration. Furthermore, if the backup system and monitoring system form the basis of the system safety justification, issues of fault coverage, latent fault scrubbing, and the hardware level of redundancy required within the backup system will also need careful treatment and consideration.

Given the above discussion, it may be argued that the current prescriptive guidance is necessary but insufficient for machine learning based applications. It is still applicable, and any software or hardware implementation of machine learning should meet the intent of the current standards. But this guidance is insufficient to accommodate the emergent and structural complexity of typical machine learning implementations. Hence, some augmentation is required.

5.4 A Path Forward

In 2014, Mike Dewalt, who was the FAA Chief Scientific and Technical Advisor for Aircraft Computer Software at that time, proposed a new Technology Independent Assurance Method (TIAM) as a potentially less prescriptive certification framework.[60] This work laid the foundation for a new FAA initiative called the Overarching Properties, that is attempting to develop additional guidance for such an approach.[61] Given that the current prescriptive guidance is insufficient, these new approaches may be applicable to accommodate the complexities of machine learning-based implementations. Machine learning certification may indeed be a great test case to assess the applicability of this emerging guidance paradigm. The Overarching Properties approach requires that an applicant for certification must demonstrate that their system meets acceptable criteria for the following basic properties:

1. **Intent –** The defined intended behavior is correct and complete with respect to the desired behavior.
2. **Correctness –** The implementation is correct with respect to its defined intended behavior under foreseeable operating conditions.
3. **Innocuity (previously termed acceptability) –** Any part of the implementation that is not required by the defined intended behavior has no unacceptable safety impact.

From the prior discussion, these properties appear to be equally applicable and very salient to the challenges of machine learning certification. However, the difficulty of arguing intent (the requirements are correct and sufficiently complete), correctness (the implementation will always perform as expected) and innocuity (the implementation is limited to intended functionality) does not decrease, though the method of justification may differ.

A related European research program (Re-Engineering and Streamlining Standards for Avionics Certification or “RESSAC”) completed in late 2017.[62] A goal of the RESSAC project was to assess the feasibility of an Overarching Property-based certification paradigm. The research team concluded that an assurance/safety case approach might be the most effective manner to support such a paradigm.

---

60 DeWalt, M., and McCormick, G., "Technology Independent Assurance Method," 2014 IEEE/AIAA 33rd Digital Avionics Systems Conference (DASC), IEEE, 2014, https://ieeexplore.ieee.org/document/6979529, retrieved June 22, 2019.

61 Holloway, C., “Understanding the Overarching Properties: First Steps,” September 2018 Draft, https://www.faa.gov/aircraft/air_cert/design_approvals/air_software/media/TC_Overarching.pdf, retrieved June 22, 2019.

62 Brown, D., “An Alternative Approach to DO 178B,” in High Integrity Software conference, 2017, https://www.slideshare.net/AdaCore/an-alternative-approach-to-do178b, retrieved June 23, 2019.

However, one challenge with the safety case approach is the justification of confidence. As illustrated by the work of Graydon and Holloway,[63] industry still lacks an accepted credible method to argue confidence claims. The lack of such foundational theory may eventually pace the insertion of new non-traditional technologies such as machine learning applications. Further research is therefore needed to understand how such confidence claims can be built and justified.

[63] Graydon, P. and Holloway, C., "An Investigation of Proposed Techniques for Quantifying Confidence in Assurance Arguments," NASA/TM–2016–219195, Safety science 92 (2017): 53-65. https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20160006526.pdf, retrieved June 23, 2019.

6 Considerations for Bounding the Behavior of ML Algorithms

6.1 Runtime Assurance Architectures

6.1.1 Runtime Assurance Definition

Runtime Assurance (RTA) may be described as the ability to ensure the safe operation of a system that contains functional components, which may not be sufficiently reliable or verified, according to current development or certification standards. RTA applies runtime verification to ensure that the system does not take unsafe actions. It steers the system into a safe state when anomalous behavior is detected. An RTA architecture has a monitor that can monitor critical safety states of functional components that may not be reliable or thoroughly verified and can switch to a reversionary system when an unsafe condition is approaching or has occurred. Unlike the functions which are being monitored, the monitor(s) and the reversionary system can be developed using existing developments standards such as RTCA DO 178C and RTCA DO 254. Figure 18 illustrates a typical runtime assurance architecture.

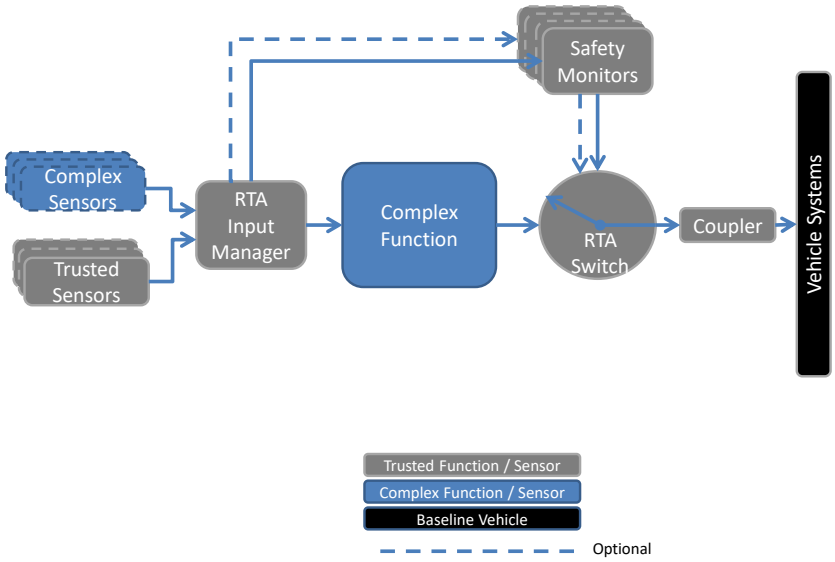

Figure 18: Typical runtime assurance architecture.[64]

In Figure 18, monitoring is an architectural mitigation against unsafe behavior of the complex function. This type of monitoring can be defined as “system level monitoring.” It is also possible to define specific monitoring directly linked with the machine learning architecture and behavior. These two types of monitoring are described in the next paragraph.

64 Hook, L.R., Skoog, M., Garland, M., Ryan, W., Sizoo, D., & Vanhoudt, J. (2018). Initial Considerations of a Multi-layered Run-time Assurance Approach to Enable Unpiloted Aircraft. 2018 IEEE Aerospace Conference, 1-11.

6.1.2 Reference Architectures

6.1.2.1 Simplex vs. Complex System-Level Monitor

The objective of system-level monitor is to define and to monitor a safe boundary of the system, regardless of machine learning implementation (same architecture as today when a system is monitored by an independent one). If the system-level monitor detects a potential safety issue for the system, the system-level monitor is able to inhibit the System Based on Machine Learning (SBML) and to continue in a safe mode (with or without aircraft limitations). This type of runtime monitor is efficient only if the aircraft function is feasible with "usual" avionics inputs/outputs. In this case, the machine learning system is used to improve efficiency or accuracy of the system but the system-level monitor is always able to check that the SBML stays in the system safe area. The system level monitor can be represented as in Figure 19.

It is important to maintain suitable segregation and independence of the monitor and backup functions, including sensors, interfaces, power, computing resources, etc. The architecture should prevent failures of the SBML components from propagating to or affecting the System Level Monitor and Backup channels. The architecture should ensure the selection switch is controlled by the trusted RTA components.

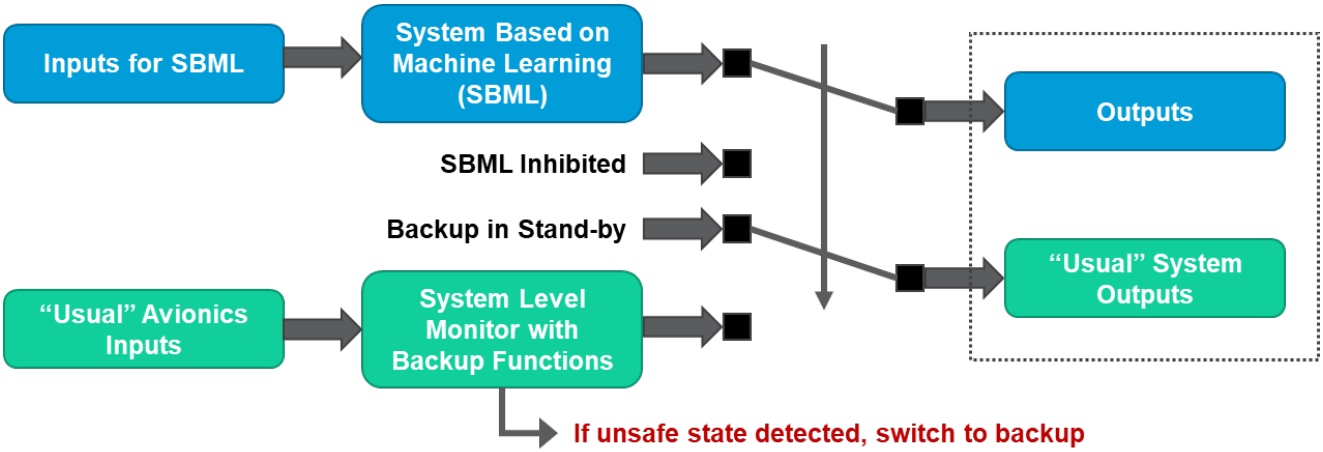

Figure 19: System level monitor.

A variant of this architecture is to use same physical outputs between the two systems. In this case, the runtime monitor is able to validate the outputs of the SBML before sending the order to the aircraft.

Nevertheless, the aerospace community would like to introduce SBML to develop new functions for more aircraft safety and efficiency. These new functions offered by machine learning technology are not feasible with current classic technology making it very difficult to use of system-level monitoring. In this case, tightly coupled runtime monitor could be used as explained in the next paragraph.

6.1.2.2 Tightly Coupled Runtime Monitor

A tightly coupled runtime monitor is a mechanism able to observe internal data of a system, and to execute dynamic verification in order to detect erroneous behavior of the system. As shown in Figure 20, the dynamic verification can be done by comparing the extracted data with data from a reference model, specification, or from previous recording during the system definition phase.

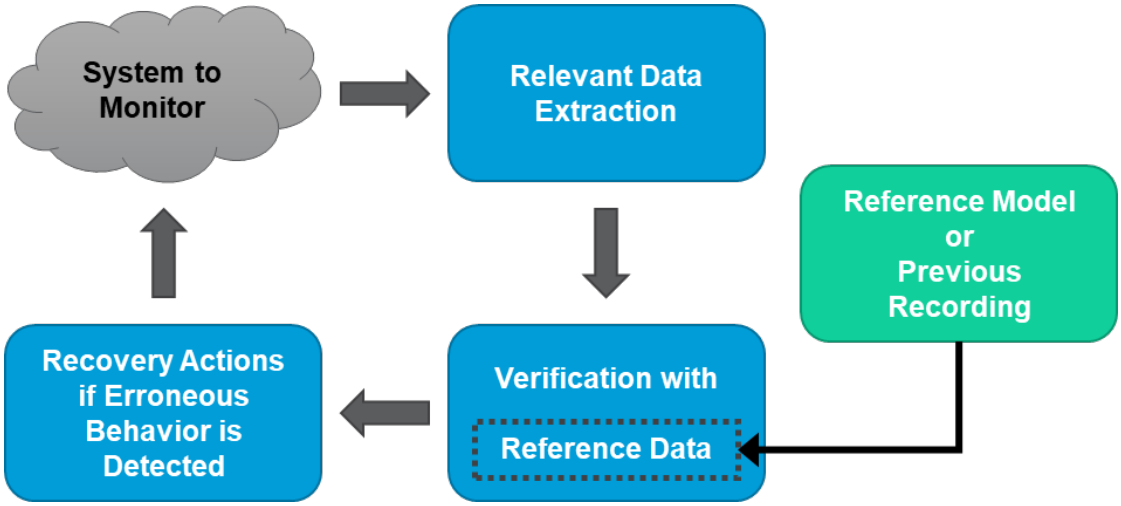

Figure 20: Reference model based assurance.

Cassar, et al. detail different techniques for runtime monitors, along with associated tools. The paper identifies a large spectrum of techniques:[65]

- **More intrusive** (completely-synchronous or "CS"), where each step of the system is verified before the execution of the next step. This technique might potentially impact the processing power needed to avoid system slow down.
- **Less intrusive** (completely-asynchronous or "CA"), where verification is done as a background task. This technique has the drawback of an increase in the time to detect an error.
- **Intermediate techniques** such as Synchronous Monitoring with Synchronous Instrumentation (SMSI), Asynchronous Monitoring with Checkpoints (AMC), and Asynchronous Monitoring with Synchronous Detections (AMSD).

Implementing these types of runtime monitors depends on the ability to identify:

- system properties that must be monitored in order to maintain system behavior within safe boundaries, and
- data in the system implementation that are representative of these properties.

The difficulty of this identification depends on the complexity of the system and the type of machine learning algorithm.

6.2 Runtime Monitor Functions and Operations

Formal verification of software where programs written in semantically well-defined languages can be mathematically proved to satisfy a logical statement specification is an established area of research in computer science that increasingly is being used in industry. Runtime Verification (RV), where specifications are checked for satisfaction at runtime can greatly increase our confidence that the machine learning application is doing what it is supposed to do or at least not doing something harmful. Like testing, RV cannot prove the absence of an error, but unlike testing, it can direct mitigating actions

[65] Cassar, I., et al., "A Survey of Runtime Monitoring Instrumentation Techniques," EPTCS 254, 2017, pp. 15-28, https://arxiv.org/abs/1708.07229, retrieved June 23, 2019.

to be performed when an error is detected. Table 3 provides definitions for some of the terminology use in RV.

Table 3: Definitions of some runtime verification terminology.

Specification	The logical definition of a correctness property.
Observability	Ability of system state to be observed during execution.
System under observation	The software/system on which runtime verification is performed.
Instrumentation	Specific software/hardware that enable the monitor to observe the system.
Trace	An execution trace of the observed system.
Monitor	System module that runs during execution and determines if a given trace violates a given specification.
Runtime Verification Framework	An RV function that, given a logical specification and possibly additional details about the system under observation, generates executable monitors including the instrumentation.
Steering problem	What to do when a specification is violated.

Specifications are mathematically precise statements of the properties that the system must satisfy. In particular, a specification must be written as a formal statement in some well-defined logic having a formal syntax and semantics (model theory). Logics that are popular for writing formal specifications include first order, higher-order logic (HOL) theorem provers (like PVS, Coq, Isabelle, HOL4, etc.), modal logics, temporal logics, stochastic logics, etc. The sort of mathematical models that engineers in aerospace are familiar with can be translated into formal specifications. Expertise with the specification formalism is usually required in order to write correct and effective specifications. The formal nature of the specification can be invaluable because it allows one to not only precisely formulate tests, but to mathematically prove properties about the specification. Yet, validation of the specifications remains essential to ensure the properties specified are the desired ones.

In general, the literature classifies specifications as one of the following three types of properties:

- **Functional correctness –** for each input it produces the expected output. For instance, for a library implementation of quicksort the specification may say given an input sequence x, the output y is a permutation of x sorted by some ordering relation. Probabilistic and randomized algorithms have specifications with a probabilistic component.
- **Safety property –** says “something bad will never happen.” This is usually defined in terms of unsafe regions of the state space.
- **Liveness –** the system makes progress. This is important in concurrency where processes may have to take turns executing critical sections of code.

Safety properties are often easier to check at runtime than functional correctness and most of the RV literature focuses on safety properties. Unfortunately, any piece of code that “checks for something” is often called a monitor. In RV, the term “monitor” means an executable realization of a formal specification. RV frameworks translate a logical specification into an executable monitor. The need to translate a logical specification into an executable monitor can restrict the choice of specification logics. For instance, there are well known algorithms for translating linear temporal logic (LTL) into efficient automata.

In RV, one can only speak about "correctness" in terms of a trace of the evolving system state and whether this trace satisfies the given specification. That is a specification defines a property of the trace. For a completely synchronous RV monitor, the trace may include internal state changes of the system under observation as well as streams of sensor data where for completely asynchronous RV monitor, a trace may only contain sensor data as the sensor data is part of the state of the evolving system. Technically speaking, given a specification of correct behavior φ and an execution trace τ, one can think of RV as checking for language inclusion $\tau \in L(\varphi)$. You can only write specifications enforcing properties on information included in a trace. Consequently, one must think about the trace when selecting a specification logic as well as when writing properties. This means that the instrumentation must be as intrusive as needed to provide the desired trace.

There are two general philosophies that are adopted in the RV community. The more intrusive approach monitors every state change. This allows one to know that some property is NEVER violated, but the executable must be instrumented so that every state change/assignment statement has some code to record the change in the trace. A lot of work has been done on the engineering behind this, especially for Java where specialized Java Virtual Machines can be built to support capturing trace information. A second approach is sampling where the state of execution is sampled at specific intervals. For hard real-time systems in particular, one can schedule the sampling in intelligent ways by scheduling when the monitor is executed. While sampling can miss some errors, it is often sufficient for physical systems. Thus, if one does not understand the trace, one cannot understand if the specification is correct and cannot know what system instrumentation is needed.

The issue of what to do when a monitor detects an error is known as the steering problem, as the RV system is expected to invoke software that "steers" the system into a safe state. The steering problem is difficult because it is application specific. In the classical Simplex architecture, there is some "conventional" control system to fall back on. In the case of satellites for example, there might be a well understood "safe mode" (point the antenna in a particular direction, reboot the system, and wait for instruction). In other systems, it might just be an alert to a human. This application-specific quality seems to have discouraged active research and publication in the area.

6.3 Limitations of Using RTA Architectures

Edsger Dijkstra's famous maxim that testing can show the presence of bugs but cannot prove their absence holds for runtime verification as well.[66] Formal proof techniques can show a program is correct for an infinite number of executions covering all possible execution paths, but runtime verification can only make assertions about the executions of the system seen thus far.

In order to ensure that an RTA architecture provides sufficient protections, the RTA architecture needs to be engineered according to conventional safety-critical avionics techniques, even if the complex function being monitored is at a lower development assurance level. All the traditional requisite safety analyses and ensuing design considerations are needed to ensure that the RTA architecture functions as intended in all foreseen environments. For instance, the RTA architecture should be fault tolerant to some number of sensor failures and analysis may need to ensure that monitors are not basing decisions on faulty sensors. An RTA architecture that does not detect and mask sensor failure will not provide much protection against one of the most common hazards in avionics. Such levels of fault tolerance will likely require that an RTA architecture incorporate a combination of redundancy and independence from the system under observation.

The RTA monitors (at system level) need a development assurance level appropriate to the failure classification of the function being monitored. For a complex, highly-coupled RTA architecture, such a

[66] See for example https://en.wikiquote.org/wiki/Edsger_W._Dijkstra

classical approach may not work. For example, consider a system where the backup channel is also the primary channel and the "complex function" is an enhanced or augmented capability (e.g., enhanced ride quality or fuel efficiency) that may have a lesser failure classification. In such a system, it must be shown that all failures of the complex function do not cause an unsafe situation, or are detected by the monitor and switched out.

Can "safety" be defined in such a way that bounds the failure mechanisms? Degraded performance may be deemed "safe." In other words, failover to a less capable backup channel may be safe, even if that backup channel is less capable.

In general, there are two primary functions of interest: 1) what triggers the monitor, and 2) the backup function. Systems will likely have many monitors each triggering a particular steering action upon detecting an error. We know from experience that events often trigger many cockpit alarms and similarly one can easily foresee concurrently executing monitors simultaneously detecting errors and directing steering. Thorough analysis needs to be conducted to prove the absence of harmful interactions.

6.4 RTA Architecture Use Cases

Having discussed a few "general concepts" in RV, the following presents more specific considerations for applying RTA to machine learning.

Consider as an illustrative use case an image classification system using a neural net for machine learning. If functional correctness is the property to be enforced, the corresponding specification may be something like "correctly classifies images 93% of the time," but it is not clear how to formally express this as a property of an observable trace. Another option is to detect undesirable behavior of the overall system. Given the current state of the art, it is not clear if it can be sufficiently determined at runtime if something is misclassified. One might be tempted to run two or three "independent" ML approaches and perform voting, but it is not clear how to ensure the level of independence necessary for such an approach to work. On the other hand, assuming the classifier is part of an "intelligent" autopilot, then from a higher-level perspective we don't care whether something is misclassified, only that the aircraft remains a "safe" distance from any object. This option can use well understood techniques, but doesn't verify the learning. It just verifies that the ML system isn't causing any bad effects.

The necessary trace depends on the specification. The trace must at least capture the input to the classifier and the output in order to monitor that a classifier "correctly" classifies an image, but may also possibly need to capture some internal state. On the other hand, it is possible that only ADS-B and basic aircraft state information need to be captured in the trace to determine that safe distances are maintained from other aircraft.

Whatever is being monitored, a "trace" of some kind will be needed. The choice of what trace to monitor needs to be well understood. The choice needs to be made based on the properties that need to be satisfied. Applying this to this use case of an image classifier used for obstacle avoidance yields some representative questions:

1. Is the trajectory the only concern, in which case detailed identification of the obstacle is not necessary?
2. If the image classifier fails to even detect an object; is there a backup detector that can be relied upon? In this case, the runtime assurance argument is made at the system level that includes the backup detector.
3. If the backup can reliably avoid the obstacle, what trace info is needed for post-event forensics to understand why the classifier missed the object?
4. How are the image classifier properties expressed?

5. What statistically significant amount of test data is required if the monitored property is statistical in nature or if the property is expressed in a suitably complex logic?
6. Can effective and efficient monitors be implemented for an image classifier?

Similar questions can be posed for a variety of other use cases. These concerns must be carefully considered during system architecting, development, and certification to ensure the efficacy of the RTA and satisfy overall system safety requirements.

7 Summary of Recommendations

7.1 Recommended Safety Objectives When Using ML in Safety-Critical Applications

This section summarizes recommendations suggested in the body of this report. The intent of this report is to provide aerospace industry stakeholders (certification authorities, ML-system developers, and integrators) with considerations that should be addressed in future guidance and policy; and recommendations for further research and development of technologies, tools, and processes useful for assuring the safety of ML-based systems safety-critical applications. Considerations are posed as directives or questions depending on the context. The recommendations are premised on:

- The context of the set of recommendations is related to artifacts and processes above the component level.
- The sequence of recommendations is based on the proposed life cycle as presented in Section 4.
- ML-based systems development will follow Means of Compliance which guide applicants and certification authorities.

7.1.1 Recommendations Related to Machine Learning Life Cycle:

1. Investigate the rationale for employing machine learning.
2. Investigate how machine learning is intended to be used. As a design aid? Using fixed runtime parameters or employing in-service updates? (Note: adaptive learning is not recommended by this report.)
3. Investigate any specific requirements for ML, for data pre-processing and for output monitoring.
4. Investigate the paradigms being considered (e.g., neural network, support vector machine, decision tree, random forest, etc...) and what is the rationale for their use.
5. Once a topology (architecture) is selected, investigate the development activities involved, such as selection of hyperparameters, functions, architecture minimization (elimination of unnecessary elements that might lead to dead code), response time, resources consumption, etc. What is the rationale for the chosen topology and how does it satisfy the corresponding set of requirements?
6. In addition to traditional system verification evidence, request verification evidence related to:
    a. Robustness criteria,
    b. Sufficiency of the range of input data to verify that the ML-system performs its intended function when subject to both “normal” and robustness requirements.
    c. Whether monitors are working correctly against normal and robustness requirements.
7. Understand the transition criteria between each phase of the machine learning development life cycle.

7.1.2 Recommendations Related to Datasets:

8. Is the dataset consistent with requirements in terms of data types and expected ranges?
9. Have criteria and processes for dataset curation been identified?

a. Criteria for dataset characteristics such as quantity of data, distribution in terms of expected usage domain, independence of input types, data segregation (training, test and verification), and elimination of outliers?

b. Criteria for data quality processes such as data curation/cleansing, dataset segregation, quality criteria?

7.1.3 Recommendations Related to Robustness

10. Robustness analysis requires intimate knowledge and understanding of the application domain requirements. Tradeoffs between robustness and the desired level of sensitivity to input data need to be understood, expressed in a quantifiable manner and considered.
11. Use of open–source/third party ML software should be taken with care to reduce vulnerability to white source attacks (even if such software is used only in design stage and never gets deployed on aircraft).
12. Robust classification can be analyzed and verified only with respect to a specific training set. Robust regression can be achieved uniformly, for an entire input domain. The training/verification set needs to be properly selected for robustness.
13. Robustness can be improved by selecting a simplified neural network architecture (e.g. a ReLU network).
14. V&V activities performed on neural networks need additional considerations. Classical V&V methods (as defined by DO-178B/C) are not sufficient for ML techniques, but still apply to the software implementation.

7.1.4 Recommendations Related to Safety Assurance

15. When using architectural mitigation strategies, such as a mix of machine learning implementations with traditionally implemented safety monitors, the analysis of the safe reversionary state or backup mode needs meticulous attention. Any reduced or simplified functionality needs to be shown sufficient for all anticipated operational scenarios. Furthermore, if the backup system and monitoring system form the basis of the system safety justification, issues of fault coverage, latent fault scrubbing, and the hardware level of redundancy required within the backup system need careful treatment and consideration.
16. The Overarching Properties require the justification and argumentation of the basic properties:
    a. Intent – The defined intended behavior is correct and complete with respect to the desired behavior
    b. Correctness – The implementation is correct with respect to its defined intended behavior under foreseeable operating conditions.
    c. Innocuity – Any part of the implementation that is not required by the defined intended behavior has no unacceptable safety impact.

7.1.5 Recommendations Related to Runtime Assurance:

17. If the system-level monitor detects a potential safety issue for the system, the system-level monitor is able to inhibit the ML-based system and to continue in a safe mode (with or without aircraft limitations).
18. According to the criticality of the function using ML, suitable segregation and independence of the monitor and backup functions should be implemented.

19. The architecture should prevent erroneous behavior of ML functions from propagating to or affecting the system level monitor and backup channels. The architecture should ensure the selection “switch” is controlled by the trusted RTA components.
20. In order to ensure that an RTA architecture provides sufficient protections, it needs to be engineered with the same safety rigor as conventional safety-critical avionics. For example, all the traditional requisite safety analyses and ensuing design considerations are needed to ensure that the RTA architecture functions as intended in all foreseen environments
21. Such levels of fault tolerance will likely require an RTA architecture incorporate a combination of redundancy and independence from the system under observation.
22. For a complex, highly-coupled RTA architecture, such a classical approach may not work. In such a system, it must be shown that all failures of the complex function do not cause an unsafe situation, or are detected by the monitor and switched out.
23. Whatever is being monitored, a “trace” of some kind will be needed. The choice of what trace to monitor needs to be well understood. The choice needs to be made based on the properties that need to be satisfied.

7.1.6 Recommendations Related to AI Interpretability:

24. Follow best practices when developing machine learning models by continually monitoring and storing metrics related to previous and current performance (e.g. recall, accuracy, etc. on training, validation, and test datasets). While such metrics do not provide direct insight into a black box implementation, in conjunction with a number of interpretable AI techniques mentioned in this document, metrics can help pin-point areas of improvement, additional training needed, and/or a lack of sufficient data in the operational domain.
25. Employ relevant interpretability techniques mentioned herein (see Appendix A and Section 3.3 on AI Interpretability) such as, but not limited to, measuring neuronal activation, saliency maps, behavioral clustering, etc. While such techniques might not directly provide sufficient evidence for certification, these methodologies enable more transparent development/training and provide insight into the functionality (strengths, weaknesses, and gaps) of a trained ML-model’s optimized values over time. AI interpretability techniques will greatly assist with design, development and debugging of AI models over the software life cycle.
26. Consider attempting to reverse engineer the trained black box into any number of white box AI systems. While performance of the newly reverse engineered white box system might not match that of the original ML model, developers can potentially intuit meaningful insights regarding the underlying system and design domain.

7.2 Recommendations for Community Standards

As stated above, one of the motivations for this project was the lack of available standards and guidance useful for developing ML-based systems in safety-critical applications. The AFE 87 PMC recommends that appropriate standards development organizations consider developing community standards that define objectives for the following machine learning processes:

- Training dataset curation,
- Machine learning training,
- Using ML safety assurance case demonstration, and
- Using simulation and modeling for assuring ML-based systems.

There are a number of relevant committees that have formed to address AI/ML in safety-critical applications. These include the SAE International G-34 Committee on Artificial Intelligence in Aviation and the EUROCAE Working Group 11 on Artificial Intelligence (now working jointly with G-34). Additionally, RTCA, Inc. has issued a call for interest in forming a new special committee on Artificial Intelligence. AVSI AFE 87 PMC members have been active in promoting establishment of these committees and intend to contribute to their efforts to promote the recommendations herein.

7.3 Recommendations for Future AVSI Research

The AFE 87 Program Management Committee recommends follow-on work focusing on working two example machine learning use cases: 1) a regression use case, and 2) a classification use case. These two use cases will encompass many of the challenges and issues identified in this report. The objective of the follow-on work is to validate the processes and objectives recommended by this report.

For each use case, the follow-on project should address the following:

1. **Work an example ML application –** The intent of this activity is to exercise the concepts herein on a representative example for each of the use cases. These examples should be simple enough to allow progress within the context of an AVSI project, but detailed enough to provide insight on the utility and completeness of the recommendations in this report.
2. **Training dataset use, management, quality factors, and processes –** This activity is intended to look specifically at the requirements on dataset used to train ML-based system used in aerospace. The PMC recommends that the project engage university researchers to examine aero-focused dataset needs. The scope should focus on the training part of an ML development life cycle. Outcomes could include definition of the characteristics of a “golden” dataset and recommendations for managing the IP aspects of datasets.
3. **ML specification –** This activity should develop specific guidelines on how to specify the “unspecifiable” (e.g. recognizing a pedestrian). This should include not only the specification of the functional behavior, but also of (formally) verifiable safety properties.
4. **ML Validation and Verification –** This AFE task should examine the use of both formal methods and testing to validate the correctness of an ML-based system and verify that it meets all safety and performance requirements.
5. **Application of Overarching Properties to ML –** This potential activity would investigate ML assurance from a systems level perspective by employing concepts being developed under the FAA sponsored Overarching Properties effort. This could include developing and demonstrating techniques for applying an assurance use case approach to argue achievement of the intent, correctness, and innocuity properties.
6. **Simulation and Modeling for ML –** the scope of this task would incorporate development of recommended requirements for the application of simulation and modeling to accelerate the learning of ML-based systems. Additionally, the AFE should develop similar requirements for the use of simulation and modeling in the validation, and possibly verification, of trained ML-based systems.

7.4 Conclusion

These recommendations offered by the members of the AVSI AFE 87 project are intended to help accelerate the development and availability of guidance and technologies useful for integrating ML-based systems in safety-critical aerospace applications. The need is urgent as AI/ML technologies are rapidly being deployed in other applications, driving down the cost of implementations and increasing the confidence in these technologies. Traditional aerospace industry stakeholders are eager

to reap the potential benefits of ML, while new applications such as unmanned systems and urban mobility are driving the desire to transition these technologies from other domains into aerospace. The AFE 87 PMC encourages cooperation among all stakeholders concerned with applying AI/ML in safety-critical applications to promote further research and activities that address the current gap in our ability to assure the safety of these technologies.

8 Glossary

Correctness	The implementation is correct with respect to its defined intended behavior, under foreseeable operating conditions.
Datasets	A set of input vectors, related to requirements in the sense that requirements express their format, margins and ranges. These sets are used to train and validate ML topologies.
Functional Correctness	For all inputs, the ML-based system produces the output expected in the context of the intended behavior.
Innocuity	Any part of the implementation that is not required by the defined intended behavior has no unacceptable safety impact.
Instrumentation	specific software/hardware that enable the monitor to observe the system.
Intent	The defined intended behavior is correct and complete with respect to the desired behavior
Labeling	A factor influencing data quality for ML solutions employing supervised learning. Algorithms learn from a dataset of examples labeled with an output variable representing the right answer.
Liveness	A specification intended to assure that the system makes progress. This is important in concurrency where processes may have to take turns in executing critical sections of code.
Monitor	Runs during execution and determines if a given trace violates a given specification.
Observability	Ability of system state to be observed during execution.
Optimization	Calculating and/or iteratively converging towards a better collection of elements (from available alternatives) providing improved representation of a target task.
Paradigm	Refers to the type of ML model (e.g., neural networks, decision trees, support vector machines, etc.) used to implement an ML-based system.
Reinforcement Learning	A machine learning paradigm in which the trained model's output (action) is iteratively optimized via a reward heuristic. Positive rewards encourage similar behaviors over time, while negative rewards discourage them.
Representativeness	A factor influencing data quality for ML solutions concerning the level to which training data contains all foreseen scenarios in which the system will be used.
Runtime Verification Framework	Given a logical specification and possibly additional details about the system under observation, an RV framework generates executable monitors including the instrumentation.
Safety Property	A specification that says "something bad will never happen." This is usually defined in terms of unsafe regions of the state space.
Specification	In the context of runtime verification, a specification is the logical definition of a correctness property.
Steering Problem	What to do when a specification is violated.

Sufficiency	A factor influencing data quality for ML solutions concerning the quanitity of data used for training. The dataset should be sufficient in quantity to train the system to meet its requirements, which depends on the problem domain.
Supervised Learning	In supervised learning, the computer algorithms are “trained” by a “training dataset” of inputs tagged with desired outputs. Supervised learning can be applied, for example, as uni- or multivariate regression, to find the best algorithmic fit to a given dataset, thereby enabling prediction of the output from a new data point.
Topology	Once a paradigm has been selected (such as neural network), the topology identifies design details such as the shape of the network, how deep it is, how many inputs it uses and how many outputs there are. Training data is then used to set weights to be used so that the outputs of the topology satisfy the requirements for the expected range of values in the n-dimensional set of inputs.
Trace	an execution trace of the observed system used in runtime verification.
Unsupervised Learning	In unsupervised learning, the input dataset is not a priori structured and the computer algorithm is used to find structure(s) in the data. Unsupervised learning can be applied, for example, as a form of data mining in unstructured datasets to reveal otherwise unknown structures and relationships in the dataset.
Validation	An engineering activity with the goal to assure that a product satisfies its intended customer use. In the context of datasets, a validation dataset is used to select the best subset of models before explicitly testing.
Verification	An engineering activity with the goal to assure that an implementation satisfies all requirements or specifications. In the context of ML datasets, a verification dataset is used to verify the behavior of the executable code once it has been integrated into the target platform/system.

Appendix A: Observability

A.1 Dimensions of Interpretability

The degree to which a black box system can be interpretable is in part determined by a number of related factors or properties of the underlying system. Several associated factors include: complexity, time limitations, global versus local interpretability, user expertise, and usability. This section provides some insight and definitions to such terms commonly found within the literature.

- **Complexity:** Hara and Hayashi identify complexity by the number of regions or parts of the model, for which boundaries are defined.[A1] Ribeiro, Singh, and Guestrin define complexity for linear models as the number of non-zero weights and as the depth of the tree for decision trees.[A2] Deng suggests that the complexity of a rule (and thus of an explanation) is measured by the length of the rule condition, defined as the number of attribute-value pairs in the condition.[A3]
- **Time Limitation:** An important aspect is the time that the user has available or is allowed to spend on understanding an explanation. Intuitively, a longer time limitation potentially enables the user to dedicate more mental resources to comprehending a system response.[A4]
- **Global and Local Interpretability:** A globally interpretable model is one that may be completely interpretable, i.e., we are able to understand the whole logic of a model and follow the entire chain of reasoning leading to all the different possible outcomes. On the other hand, local interpretability only affords reasons for a specific decision versus the entire system.[A5]
- **Nature of User Expertise:** Users of a model may have different background knowledge and relevant domain experience. For instance, the mechanisms for interpretability are likely and necessarily different for policy-makers and technology consumers compared to compliance and safety engineers or data scientists.[A4]
- **Usability:** Usable models provide information that assist users in accomplishing a task with awareness. A usable technology typically has an explicit or more comprehendible intended use case. Through understanding intended use, people can gain insight into model behaviors. [A4]

A.2 Interpretability Definitions and Related Terms

It's important to discuss the meaning of interpretability used within this document and in general within the broader community. There are a number of related terminologies and factors regarding techniques which aid in exposing and interpreting black box systems. However, a consensus on terminologies is

---

A1 Hara, S. and Hayashi, K., "Making Tree Ensembles Interpretable," June 17, 2016. https://arxiv.org/abs/1606.05390, retrieved June 23, 2019.

A2 Ribeiro, M., Singh, S., and Guestrin, C., "Why Should I Trust You?: Explaining the Predictions of Any Classifier," Proceedings of the 22nd ACMSIGKDD International Conference on Knowledge Discovery and Data Mining, pages 1135–1144, ACM, 2016, https://arxiv.org/pdf/1602.04938.pdf, retrieved Apr. 18, 2019.

A3 Deng, H., "Interpreting Tree Ensembles with in Trees," 2016 ICML Workshop on Human Interpretability in Machine Learning (WHI 2016), New York, NY, USA, https://arxiv.org/pdf/1606.05390.pdf, retrieved June 23, 2019.

A4 Guidotti, R., et al., "A Survey of Methods for Explaining Black Box Models," ACM computing surveys (CSUR) 51.5 (2018): 93, https://arxiv.org/abs/1802.01933, retrieved April 6, 2019.

A5 Lipton, Z., "The Mythos of Model Interpretability," 2016 ICML Workshop on Human Interpretability in Machine Learning (WHI 2016), New York, NY, USA, https://arxiv.org/pdf/1606.03490v3.pdf, retrieved April 6, 2019.

not concrete within the literature. Some suggest model interpretability as a remedy to understanding a black box system, but few articulate precisely what interpretability means or why it is important.[A5]

While most of the literature comes from machine learning and data mining communities, each community addresses the problem from a different perspective and associates different meanings. These various perspectives, in addition to the already intangible nature of interpretability, observability, trust, visibility, etc. lead to difficulties in forming concrete terminology.

This section attempts to provide insight into commonly used terms and definitions. Within the literature there appears a number of terms related to the same underlying concept of AI interpretability. In data mining and machine learning, interpretability is defined as the ability to explain or to provide the meaning in understandable terms to a human. Explainable AI (XAI) is probably the most commonly used term these days, having been coined after a DARPA project of the same name.[A6] Additionally, interpretability is also at times synonymous with terms such comprehensibility.[A4]

Further, there are a number of interrelated terms associated with AI interpretability. Some more commonly associated terms are observability, trust, fairness and privacy. Transparency, when not used as a direct synonym, can also be utilized as a related term to interpretable AI:

- **Observability** is the ability of the tester to observe the state of the system to determine whether a test passed or failed.[A7]
- **Transparency** is defined as enabling an "operator's comprehension about an intelligent agent's intent, performance, future plans, and reasoning process."[A8]

Additionally, it is important to recognize that the exact definitions or implications of interpretability change through the perspective and operational level at which one is trying to diagnose interpretable designs. Interpretable AI discussed at the analytical level focusses on engendering insight and transparency into the underlying logic of the model and algorithm. There are a number of different perspectives and levels at which concepts of interpretability can be applied. This is discussed in a subsequent section.

A.3 Logical Layers of Interpretability

In discussing interpretable systems, it is likewise important to understand the distinction between different types of comprehensible AI approaches/techniques. AI interpretability is not a one size fits all solution space, but can be further characterized by the approach to which developers and researchers attempt to expose the black box. Further there are multiple layers to comprehensible AI based on the perspective in which the systems are utilized. By way of example, a comprehensible AI system according to an AI researcher will vary greatly compared to an unfamiliar end user. This section briefly discusses the decomposition of comprehensible AI based on perspective.

A6 See https://www.darpa.mil/program/explainable-artificial-intelligence

A7 Koopman, P., and Wagner, M., "Challenges in Autonomous Vehicle Testing and Validation," SAE International Journal of Transportation Safety, 4.1 (2016): 15-24, https://users.ece.cmu.edu/~koopman/pubs/koopman16_sae_autonomous_validation.pdf, retrieved June 23, 2019.

A8 Lyons, J., et al., "Certifiable Trust in Autonomous Systems: Making the Intractable Tangible", Artificial Intelligence Magazine, 38, (3), pp. 37-49, https://www.aaai.org/ojs/index.php/aimagazine/issue/view/219, retrieved May 21, 2019.

A.3.1 Technological Decomposition of Interpretability

There are numerous technical approaches to generating transparent models and providing insight into a black box system. Thus, it is helpful to compose the various technical approaches into a common methodology or set of distinct problem categories. This section highlights the general approaches researchers take toward AI Interpretability from an analytical perspective.

In general, the common technical approaches toward development of a transparent system can be categorized into four unique types: the model explanation problem, black box outcome explanation problem, black box inspection problem and transparent box design problem.[A4] Figure A-1 provides a visual representation of the four analytical problem categories. In each of the cases there are shared common modules. The learner module can be understood as a neural network or some other machine learned technique prior to and during optimization (via training on the desired dataset). The predictor is a trained system utilized for generating outputs provided some input.

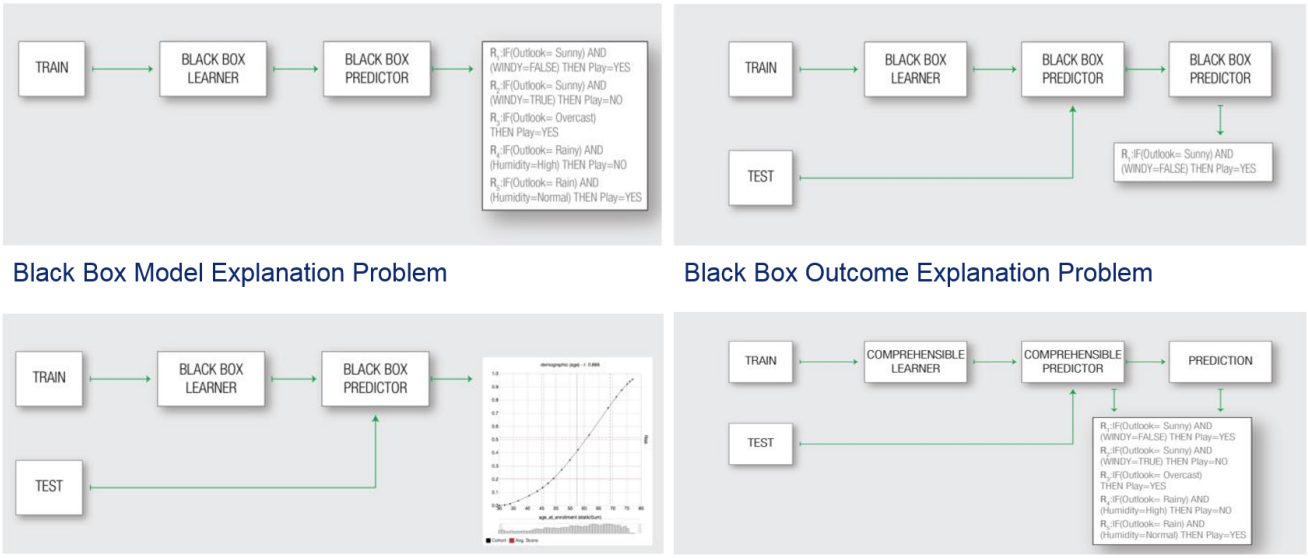

Figure A-1: Classifying the different approaches to exposing the black box.[A4]

Each category attempts to address the problem of AI Interpretability through different mechanisms varying by both technique and the specific aspects of the model exposed.

- The Black Box Model Explanation provides a globally interpretable AI system by mimicking the behaviors of the black box system via reverse engineering. An example of this use case is training a neural network on data in order to then train a more interpretable system to reproduce the neural net output, such as a regression model with hand crafted features or a Bayesian model.
- The Outcome Explanation provides a locally interpretable AI system by generating a human understandable explanation for the black box output, in addition to the output itself. Such a system might provide textual, visual or verbal explanation for the prediction or output behavior.
- The Black Box Inspection Problem explains the predictions by analyzing patterns in behaviors. This is typically accomplished through graphical and statistical means. An example of techniques

used to explain the black box in this case are sensitivity analysis, partial dependence plots and neuronal activation in the case of Deep Learning.

- The Transparent Box Design is an inherently global and locally interpretable model. Such a system might use dynamic rules-based logic, or decision trees.

The technical approaches can be further decomposed through the technique used to explain the black box system (explanator), the type of black box system, the data used by the underlying system (textual, tabular, imagery), and the generality of the explanator to name a few. Table A-1 provides a summary of the various features used to identify different technical approaches for categorizing black box interpretability.

Table A-1: Features utilized to decompose the technical approaches.[A4]

Feature	Description
Problem	Model Explanation, Outcome Explanation, Black Box Inspection, Transparent Design
Explanator	**DT** – Decision Tree, **DR** – Decision Rules, **FI** – Feature Importance, **SM** – Saliency Masks, **SA** – Sensitivity Analysis, **PDP** – Partial Dependence Plot, **NA** – Neuronal Activation, **PS** – Prototype Selection
Black Box	**NN** – Neural Network, **TE** – Tree Ensemble, **SVM** – Support Vector Machines, **DNN** – Deep Neural Network, **AGN** – AGNostic Black Box
Data Type	**TAB** – TABular, **IMG** – IMaGe, **TXT** – TeXT, **ANY** – ANY type of Data

As a final note for this section, it will be important for the aviation industry to further identify how these general problem categories address models of interest to them.

A.3.2 General Decomposition of Interpretability

From an even broader perspective, AI Interpretability can be further categorized according to the system level at which interpretability is applied. The previous section namely discussed, the analytical transparency factor. However, there are several other levels at which interpretability is relevant: the intentional, environmental, task, analytic, and teamwork models.[A8]

At a high level, the different models (Intentional, Environment, Task, Analytic, and Teamwork) each aim to provide specific transparency objectives. Table A-2 describes each of these models at a high level. Note, at each level different perspectives of the Designer, Tester, and User need to be accommodated. This will likely require specific techniques to address the unique challenges, concerns and levels of experience/familiarity with the underlying system each perspective provides.

Table A-2: General transparency levels to consider within an autonomous system.[A8]

Transparency Factors/Levels	Description
Intentional Model	High level purpose, method/style of interactions, goal structure overview, and social/moral intentions.
Environment Model	Communicates systems awareness of environmental conditions, constra ints and task related limitations.
Task Model	Regards system goals and real time progress toward goals.
Analytics Model	Regards how the system works, calculations and algorithms used, and explanations of why a technical error might occur.
Teamwork Model	Concerns the dynamics between human and autonomous teammates, as well as autonomous system to autonomous system cooperation.

A.4 Interpretability Use Cases

Explainable AI has multiple use cases and potential applications. One such broad use case could be utilization as part of a machine learning assurance case. Explainable techniques might provide needed justification and/or evidence for properties of performance. What follows are several examples of use cases for interpretable AI, which have practical applications for the use of machine learning within the aviation domain. These use cases, of which only a few are discussed here, include: justification via visual evidence, interpretability as a debugging tool, self-reporting interpretable systems, self-correcting interpretable systems, extending the training set for enhanced generalizability and robustness, and physics-guided ML. Visual evidence to justify performance, self-correcting interpretable systems, and physics guided deep learning are discussed below.

A.4.1 Visual Evidence Justifying Performance

A common technique used for explaining the predictions of Deep Neural Networks is through visualization of the features contributing to a particular prediction/model output. Regardless of the technique, the ultimate objective is to point toward the features that most strongly led to the acquired output. Visual evidence can be an intuitive mechanism, both for developers and potentially for certifying authorities, to better understand the internal logic of a black box system. While this technique alone does not provide sufficient information for an assurance claim, it can aid in developing trust of system functionality. As shown in Figure A-2, several different visualization techniques are used to expose a neural network by highlighting areas of an image that contributed to output text (left)[A9] and self-driving car control outputs (right).[A10] Nvidia is using such techniques during autonomous driving to provide additional validation and assist in passenger trust. The visualization technique helps confirm that the system is operating in conformance with expectations by attending to lane boundaries and other vehicles within the environment.

[A9] Xu, K., et al., "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention," http://arxiv.org/pdf/1502.03044v3. Retrieved June 23, 2019.

[A10] M. Bojarski, et al., "Explaining how a deep neural network trained with end-to-end learning steers a car," https://arxiv.org/abs/1704.07911 (2017).

Figure A-2: Exposing a neural network by highlighting areas of an image.[A9] [A10]

A.4.2 Self-Correcting Interpretable Systems

Self-correcting Interpretable systems close the loop between reporting environmental features driving model output, and the causality of such features. A self-correcting ML system reports the justification for actions in an interpretable way and corrects those justifications based on their causality. Such a system goes beyond the simple cycle of optimizing machine learning accuracy. It does so by optimizing the human interpretable evidence the model uses to justify its predictions or behaviors.

In the automotive example, the causal attention-based process could help verify that a steering command causing a lane change was determined through consideration of relevant environmental factors, such as the cars or pedestrians in nearby lanes. The underlying model and/or the interpretable heat map could be updated to reflect this newly verified information. Further, if the system was not attending to appropriate aspects of an image then this could trigger a more focused ML training regimen to remedy such shortcoming within particular scenarios.

Kim and Canny illustrate a mechanism for defining attention-based heat maps via causal filtering, as shown in Figure A-3 below.[A11] The top left image identifies the causal versus spurious (non-contributing) heat map blobs. The text and bottom image highlight the process for identifying and refining the heat maps based on causal regions of attention.

[A11] Kim, Jinkyu; Canny, John, “Interpretable Learning for Self-Driving Cars by Visualizing Causal Attention,” http://arxiv.org/pdf/1703.10631v1 (2017).

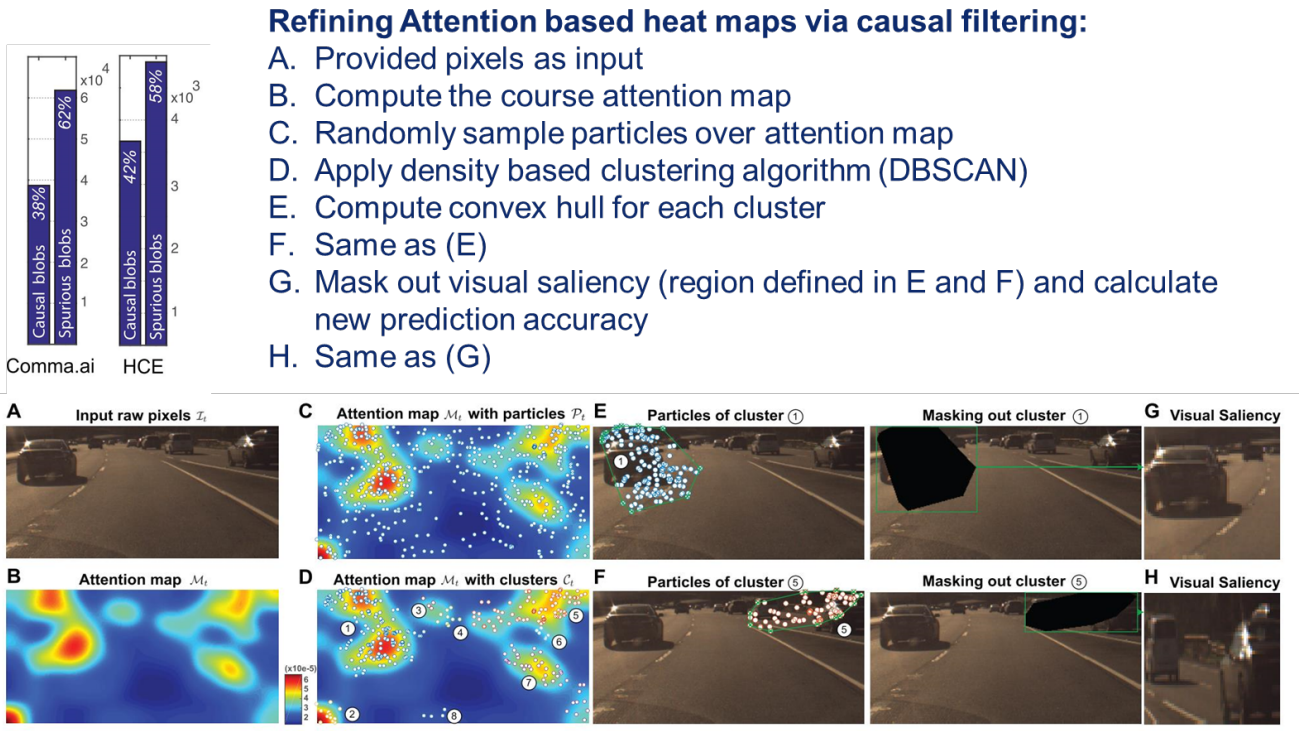

Figure A-3: Mechanism for defining attention-based heat maps via causal filtering.[A11]

A.4.3 Physics Guided Deep Learning

The ability to enforce hard constraints on deep learning systems would make them more deterministic and interpretable, thereby aiding the certification process. Aerospace applications often require the modeling of physics processes and phenomena. There's a spectrum across which these models are based on closed form equations, approximate models, all the way to pure data-based forecasting methods. Whether deep learning is useful is usually based on the accuracy and computational needs of less complex models (e.g. physics based). Methods for constraining DL systems offer varying degrees of interpretability.

Traditional physical models are explicitly constrained, while data driven models are implicitly constrained by the observed phenomenon in the training data. Common types of constraints are conservation laws and rule-based constraints. In addition, we often impose engineering constraints and specifications that are not reflected in experimental data. Generally speaking, DL systems aren't amenable to the incorporation of hard and rule-based constraints though investigations into this area are beginning to appear.[A12, A13, A14] A primary reason for this is the incompatibility of the stochastic

[A12] Ravi, S., Dinh, T., Lokhande, V., Singh, V., "Constrained Deep Learning using Conditional Gradient and Applications in Computer Vision," March 17, 2018, https://arxiv.org/abs/1803.06453, retrieved April 20, 2019.

[A13] Márquez-Neila, P., Salzmann, M., Fua, P., "Imposing Hard Constraints on Deep Networks: Promises and Limitations," June 7,2017. https://arxiv.org/pdf/1706.02025, retrieved April 20, 2019.

[A14] Karpatne, A., "How Can Physics Inform Deep Learning Methods in Scientific Problems," University of Minnesota, 2017. https://dl4physicalsciences.github.io/files/nips_dlps_2017_slides_karpatne.pdf, retrieved April 20, 2019.

gradient descent (SGD) algorithm used to optimize the objective function for most DL systems with rule-based constraints.

The most common approaches for incorporating physical constraints on DL models include:

- **Penalize the fitness function:** Placing a penalty in the fitness function to discourage solutions with unwanted characteristics is common across all machine learning methods. This is useful for constraining specific known model characteristics that are measurable in the data. This is not a *provable* constraint, as may be required for certification. The penalty may discourage exploration of feasible solutions near the regions of penalty.
- **Model the errors:** Modeling as much as possible with classical methods and using DL to model the difference between the model and observed data is often seen as a logical division of labor. This approach retains the explainability of the original model while acknowledging there may be observed nonlinear complexities that are beyond the model, but predictable. This is common in sensor data from an LRU that is part of a much larger coupled system. The example shown in Figure A-4 below illustrates an observed vibration signal of an engine is decomposed into the expected signal per physics-based modeling (red trace) and the remaining "noise" (pink trace). The noise is the information unexplainable by the model, but often contains additional useful information on complex and sometimes nonlinear behavior that is accessible via deep learning models.

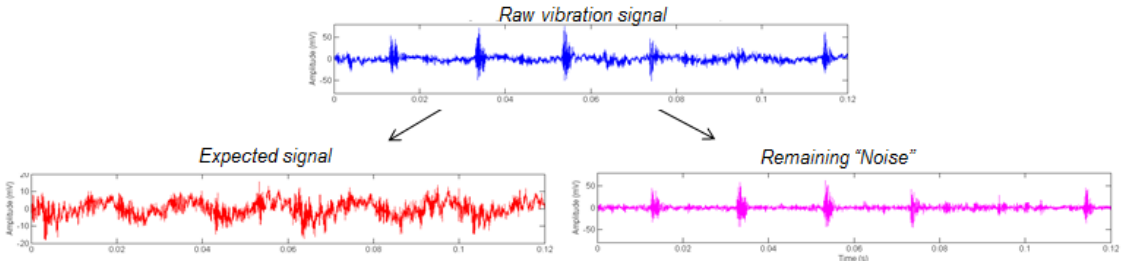

Figure A-4: A raw vibration signal decomposed into expected and "noise" components.

- **Model observed and simulated data simultaneously:** Including observed behavior along with expected behavior is another way of imbuing a DL model with constraints. Imagine having both sensor data from an engine along with the values predicted by a physics-based model. Having learned the relative properties and ranges of the inputs, the model will learn implicit constraints of possible outputs. This is not a hard constraint and the method requires foreknowledge of nominal conditions for the system. Figure A-5 shows a deep auto-encoder network as an example. In this case the model uses observed and theoretical values of temperature, pressure and vibration to learn a model to predict them. After training, even if a temperature sensor failed, its predicted value is more likely to reflect a physically likely value.

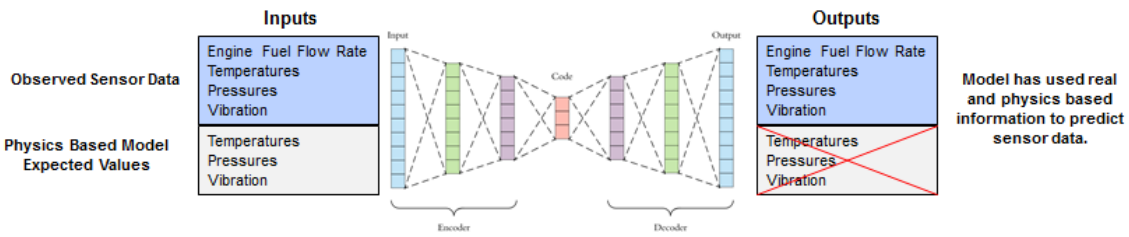

Figure A-5: A deep auto-encoder network being trained.

Inputs to the model include both sensor data and expected sensor readings from physics based models. The real and theoretical information are used to predict just the sensor data. The features learned will reflect both sources and be more robust if sensors deteriorate. This provides proxy outputs for bad sensors and an anomaly detection method.