DOT/FAA/TC-26/2

Federal Aviation Administration
William J. Hughes Technical Center
Aviation Research Division
Atlantic City International Airport
New Jersey 08405

Safety Assurance for Artificial Intelligence/Machine Learning in Safety Critical Airborne Systems

June 2026

Final report

U.S. Department of Transportation
Federal Aviation Administration

NOTICE

This document is disseminated under the sponsorship of the U.S. Department of Transportation in the interest of information exchange. The U.S. Government assumes no liability for the contents or use thereof. The U.S. Government does not endorse products or manufacturers. Trade or manufacturers' names appear herein solely because they are considered essential to the objective of this report. The findings and conclusions in this report are those of the author(s) and do not necessarily represent the views of the funding agency. This document does not constitute FAA policy. Consult the FAA sponsoring organization listed on the Technical Documentation page as to its use.

This report is available at the Federal Aviation Administration William J. Hughes Technical Center's Full-Text Technical Reports page: actlibrary.tc.faa.gov in Adobe Acrobat portable document format (PDF).

Technical Report Documentation Page

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
DOT/FAA/TC26/2		

4. Title and Subtitle	5. Report Date
Safety Assurance for Artificial Intelligence/ Machine Learning in Safety Critical Airborne Systems	June 2026
	6. Performing Organization Cage Code: 4JTJ8
7. Author(s): Prashin Sharma, Chad McFarland, Matthew Jahn	8. Performing Organization Report No.: ETT-EM-24-00477
9. Performing Organization Name and Address: The Boeing Company, 499 Boeing Boulevard, Huntsville, AL 35824 United States	10. Work Unit No. (TRAIS)
	11. Contract or Grant No.: 692M15-21-T-00035
12. Sponsoring Agency Name and Address: FAA William J. Hughes Technical Center, Aviation Research Division, Atlantic City International Airport, New Jersey 08405	13. Type of Report and Period Covered: Final Report 9/2021 – 9/2024
	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

This report addresses the challenges of integrating artificial intelligence (AI) and machine learning (ML) technologies into aviation while ensuring safety and reliability. AI/ML applications in areas such as autonomous flight control and collision avoidance show great potential, but there is uncertainty around how to employ them and certify them confidently. This report evaluates current verification and validation (V&V) methods and explores new approaches to safety assurance for AI/ML-enabled systems. The project assessed AI/ML assurance techniques, implementation frameworks, and risk mitigation strategies, including run-time assurance (RTA) and contingency management (CM). It highlights the need for improved methods to manage uncertainties in AI/ML models. Recommendations include the development of new AI-specific standards and certification processes, along with continued research into AI/ML safety, particularly in human-AI interaction and real-time monitoring. The findings emphasize that while AI/ML technologies have transformative potential for aviation, significant advancements in safety assurance frameworks and certification processes are required to support their safe and effective integration into safety-critical systems.

17. Key Words	18. Distribution Statement
Artificial Intelligence -- Machine Learning, Artificial Intelligence -- Safety-Critical Systems, Verification & Validation, Reinforcement Learning, Risk Mitigation, Uncertainty Quantification, Contingency Management, Run-Time Assurance	This document is available to the U.S. public through the National Technical Information Service (NTIS), Springfield, Virginia 22161. This document is also available from the Federal Aviation Administration William J. Hughes Technical Center at actlibrary.tc.faa.gov.

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	211	

Acknowledgements

We would like to acknowledge Dr. Huafeng Yu's immense contributions for initiating and leading the project in its initial years.

We would also like extend our sincere thanks for contributions of co-authors whose collaboration and support led to successful completion of the project.

Members from Boeing include Dr. Hieu Ngyuen, Dr. Denis Osipychev, Dr. Dragos D. Margineantu, Gordon L Putsche, Dr. Joshua D Kalin, Joshua Neighbor, Todd C Grossaint, Murali Rangarajan, Dr. Michael B Schwiesow, Dr. Amani Alonazi, Paul Stanley and Balram Kandoria.

Special thanks to Dr. Yuning He, Dr. Daniel Hulse, Dr. Lukman Irshad and Dr. Guillaume Brat from the RSE group at NASA Ames for their support.

Contents

1 Introduction 1

1.1 AI/ML techniques in autonomous systems 2

1.1.1 Autonomous functions 2

1.1.2 Typical AI/ML algorithms 3

1.1.3 AI standardization efforts 4

1.2 Tasks 4

2 Assessment of AI/ ML V&V, assurance, and implementation technologies 6

2.1 Assessment approach 6

2.1.1 Resources for technical gaps 8

2.1.2 Assurance techniques 8

2.1.3 Assurance assessment 10

2.2 Safety assurance in reinforcement learning 15

2.2.1 Core concepts in RL 16

2.2.2 Deep RL 18

2.2.3 Review of safety assurance in RL 21

2.2.4 Robustness analysis of a collision avoidance neural network 34

2.3 Reproducibility issues of PyTorch-trained models 46

2.3.1 Experiments 47

2.3.2 Parameter variability 48

2.3.3 Performance variability 48

2.3.4 Conclusion 49

2.4 Evaluation of TensorFlow ML deployment pipeline 49

2.4.1 TensorFlow deployment pipeline 50

2.4.2 Case study 55

2.4.3 Summary 61

2.4.4 Conclusion 63

2.5 Uncertainty management in ML implementations 63

2.5.1 Uncertainty in ML-based systems 64
2.5.2 Uncertainty management in learned models 69
2.5.3 Case study 79
2.5.4 Uncertainty management challenges and recommended practices 96
2.6 Safety assessment of ML-enabled systems 97
2.6.1 Review of safety assessment methods for ML-enabled systems 98
2.6.2 Fault modeling design tool (fmdtools) 100
2.6.3 Conclusion 103
3 Risk mitigation approaches in AI/ML systems 105
3.1 Run-time assurance and contingency management-based approach 106
3.1.1 Review 106
3.1.2 Generic AI RTA architecture 115
3.1.3 Generic CM architecture 118
3.1.4 Integrated AI RTA and CM 121
3.1.5 Case study: detect and avoid with ML-enabled system 123
3.1.6 Conclusion 126
3.2 Operational design domain-based risk mitigation of ML-enabled systems 126
3.2.1 Operational design domain definitions 126
3.2.2 Risk mitigation based on ODD 131
3.2.3 Conclusion 133
3.3 Human factor considerations for AI/ML-enabled systems 134
3.3.1 Review of automation levels 134
3.3.2 Applications involving Human-AI interactions in aviation 137
3.3.3 Challenges in Human-AI autonomy 139
3.3.4 Conclusion 146
3.4 Introduction for recommendations 146
3.4.1 Scope 147
3.4.2 Machine learning model development process 147

3.4.3 Aerospace standards 154
3.4.4 Recommendations 158
3.4.5 Challenges 164
4 Conclusion 166
5 References 167
A Comparison of different methods for uncertainty quantification A-1

Figures

Figure 1. Main technical areas and tasks to achieve project objectives 5
Figure 2. Approach to assess AI/ML assurance techniques 7
Figure 3. RL Agent interaction with the environment 16
Figure 4. A non-exhaustive taxonomy of RL algorithms 19
Figure 5. DQN algorithm with experience replay 20
Figure 6. PPO-clip algorithm 21
Figure 7. Number of papers studied on safety verification in reinforcement learning implementations 23
Figure 8. Venue and affiliation of paper reviewed 23
Figure 9. Environment setup for aircraft encounter 34
Figure 10. Collision avoidance NN training process diagram 37
Figure 11. Learning curve for different seed initialization 37
Figure 12. Trained collision avoidance neural network 39
Figure 13. High-level system architecture of SysAI and SuT simulation reports 41
Figure 14. All collision scenarios(left), converge from left (middle), converge from right (right) 44
Figure 15. (Left) Success and failure regions for noisy command; (Right) Success and failure regions defined by variation in speed and control command 45
Figure 16. TensorFlow ML deployment pipeline 50
Figure 17. An example subgraph formation stages using TensorFlow-TensorRT Optimization. 55
Figure 18. General categorization of deep learning based object detection meta-architectures... 56
Figure 19. YOLOv4 object detector architecture 57
Figure 20. Small section of YOLOv4 after TFLite quantization optimization 59
Figure 21. mAP and inference speed for different YOLOv4 quantized models on a Dell Workstation 59
Figure 22. Size of different quantized versions of YOLOv4 60
Figure 23. mAP and inference speed for different YOLOv4 quantized models on Jetson Nano. 61
Figure 24. Various aspects of uncertainty management in learned models 64
Figure 25. Methods for uncertainty management during different design phases of learned model 69
Figure 26. Workflow for conformal prediction of a generic ML model 76
Figure 27. Data used for case study 81
Figure 28. Confusion matrix of TinyVGG trained on Fashion MNIST dataset 82
Figure 29. Prediction score from baseline model, Prediction | True Label with p = score 83

Figure 30. Reliability diagram for baseline model 84
Figure 31. Visualization of ensemble predictions for in-distribution data, Predicted Label| True Label with p=mean score, s = std 85
Figure 32. Visualization of ensemble predictions for distribution-shift data, Predicted Label| True Label with p=mean score, s = std 86
Figure 33. Visualization of ensemble predictions for out-of-distribution data, Predicted Label| True Label w/ p=mean, s = std 87
Figure 34. Visualization of conformal prediction set for in-distribution data, True label (prediction set) 88
Figure 35. Visualization of conformal prediction set for distribution-shift data, True label | [prediction set] 89
Figure 36. Posterior Predictive Distribution of a T-shirt being classified as other objects for learned model 93
Figure 37. Sample image of a utility vehicle on a runway detected on a landing approach 94
Figure 38. Synthetic data of TRTA with respect to the detection of a particular object 95
Figure 39. Layered complexity of uncertainty 97
Figure 40. Object categorization based on object criticality 101
Figure 41. (a) OOD data; (b) Physical adversarial perturbations; (c) Class prediction confidence reduction due to image distortion 106
Figure 42. RTA architectures for reinforcement learning 107
Figure 43. ASTM RTA architecture 108
Figure 44. TaxiNet demonstration system with RTA architecture 109
Figure 45. EVAA RTA network architecture 110
Figure 46. UAS CM architecture 111
Figure 47. Intelligent contingency manager 112
Figure 48. Contingency planner architecture and procedure developing lost link contingency plan 114
Figure 49. Functional architecture for AI RTA 115
Figure 50. Functional Architecture for AI/ML Contingency Manager 118
Figure 51. Functional architecture integrating RTA and CM 122
Figure 52. Integrated RTA and CM for DAA application with perception sensors 124
Figure 53. ODD characterization 128
Figure 54. *Level 2 Human-AI Teaming as proposed in EASA Concept paper Issue 2* 135
Figure 55. Levels of automation for categories of independent function 136
Figure 56. A tree-based model on factors influencing trust in AI-based system 140
Figure 57. Reconciliation of automation error frameworks 145

Figure 58. ML model development flow diagram 148

Tables

Table 1. Typical AI/ML algorithms used in autonomous systems 3
Table 2. A summary of machine learning testing methods, extracted from (J. Zhang, 2020)..... 9
Table 3. A summary of machine learning verification methods..... 10
Table 4. Sample set of verification properties for ML..... 12
Table 5. Sample set of properties related to testing 14
Table 6. RL agent training performance metrics 38
Table 7. Parameter variations statistics..... 48
Table 8. Performance variability in different models 48
Table 9. Summary of various quantization techniques in TensorFlow Lite 52
Table 10. Summary of parameter conversion for various quantization options in TensorFlow Lite 53
Table 11. System Configuration 58
Table 12. Benchmarking summary of YOLOv4 post-training quantized models 62
Table 13. TinyVGG model summary 81
Table 14. Comparison between baseline model and voting ensemble 85
Table 15. Confusion matrix 91
Table 16. Posterior predictive probability values 92
Table 17. Severity classification defined for fmdtool analysis..... 102
Table 18. Comparative description of AI RTA and CM 121
Table 19. Non-exhuastive list of hyperparameters and their effect on models 150
Table 20. Metrics for assessing performance of classification models 151
Table 21. Performance measures for robustness evaluation of neural networks..... 153

Acronyms

Acronym	Definition
ACAS	Airborne Collision Avoidance System
ACT	Autonomous Centerline Tracking
ADS	Automated Driving System
AI	Artificial Intelligence
AUROC	Area Under Curve Receiver Operating Curve
AV	Autonomous Vehicle
BoF	Bag of Freebies
BoS	Bag of Specials
BRSL	Black-box Reachability-based Safety Layer
CBF	Control Barrier Function
CDL	Contingency Decision Logic
CES	Contingency Execution Switch
CLF	Control Lyapunov Function
CM	Contingency Management
CNN	Convolution Neural Network
COLI	Cost of Lost Information
CPG	Contingency Plan Generator
CPU	Central Processing Unit
D2RL	Dense Deep-Reinforcement-Learning
DAL	Development Assurance Level
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q-Networks
DRL	Deep Reinforcement Learning
ECE	Expected Calibration Error
EO	Electro-optical
EVAA RTA	Expandable Variable Autonomy Run-Time Assurance Network
FAA	Federal Aviation Administration
FMEA	Failure Mode Effect Analysis
FRS	Forward Reachable Set
GPU	Graphics Processing Unit
HCD	Human-Centered Design

IoU	Intersection of Union
IRP	Incident Response Plan
LEC	Learning-enabled Component
LIDAR	Light Detection and Ranging
mAP	Mean Average Precision
MC	Monte-Carlo
MCMC	Markov Chain Monte Carlo
ML	Machine Learning
ODA	Organization Designation Authorization
ODC	Object Detection and Classification
ODD	Operational Design Domain
OOD	Out Of Distribution/Out of Domain
OP	Overarching Properties
PAN	Path Aggregation Network
PPO	Proximal Policy Optimization
PPP	Preparedness and Prevention Plan
QPs	Quadratic Programs
RL	Reinforcement Learning
RMP	Risk Management Plan
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
RRP	Rescue and Recovery Plan
RTA	Run-Time Assurance
RTS	Reachability-Based Trajectory Safeguard
SFHA	System Functional Hazard Analysis
SPP	Spatial Pyramid Pooling
SuT	System under Test
TD	Temporal Difference
TF	TensorFlow
TFLite	TensorFlow Lite
TF-TRT	TensorFlow-TensorRT
TRPO	Trust Region Policy Optimization
TRTA	Time Remaining To Act
UAS	Unmanned Aerial Systems
UAM	Urban Air Mobility

UAV	Uncrewed Aerial Vehicle
UQ	Uncertainty Quantification
V&V	Verification and Validation
WocaR-RL	Worst-case-aware Robust RL

Executive summary

This report provides a detailed analysis of the integration of artificial intelligence (AI) and machine learning (ML) technologies into safety-critical airborne systems, emphasizing the importance of ensuring safety and reliability. AI/ML technologies have demonstrated their potential to enhance operational efficiency and autonomy in areas such as flight control, navigation, and collision avoidance. However, the inherent complexity of these systems, particularly regarding reliability, and verifiability, presents significant challenges in their certification and deployment. Regulatory agencies such as the Federal Aviation Administration (FAA) require stringent safety assurance frameworks to support the integration of AI/ML into aviation.

This project had four primary objectives: assessing AI/ML assurance techniques in safety-critical applications; evaluating AI/ML implementation methods with a focus on verification and validation (V&V); exploring risk mitigation strategies; and making recommendations for future standards, guidance, and certification in aviation. The assessment of V&V techniques identified critical gaps, as traditional verification approaches are insufficient for addressing the complexities introduced by AI/ML systems, which rely heavily on data-driven models rather than conventional rule-based software. While techniques such as statistical testing, formal verification, and simulation are foundational, they must be adapted to accommodate the unique characteristics of AI/ML, including neural networks and reinforcement learning (RL) algorithms.

This report focuses on RL, a prominent AI/ML technique that allows systems to learn optimal actions through interaction with the environment. The application of RL in safety-critical contexts, such as collision avoidance in autonomous aircraft, presents unique challenges. A key finding in the report is the need for effective uncertainty management in AI/ML systems. AI/ML models, especially in dynamic and safety-critical environments, must account for variability in both data and operational contexts. Techniques such as conformal prediction, ensemble learning, and Bayesian inference are explored as potential methods for improving the robustness and reliability of AI/ML systems.

The report also examines various risk mitigation strategies, with a focus on run-time assurance (RTA) and contingency management (CM) systems. These approaches are designed to ensure that AI/ML systems operate safely in real time, with RTA providing ongoing monitoring to keep operations within defined safety boundaries, and CM intervening when necessary to prevent unsafe behavior or system failures. New certification criteria that account for the specific

challenges of AI/ML are essential for their successful integration into safety-critical environments, particularly in aviation.

The report concludes with several recommendations for advancing AI/ML safety assurance in aviation: formalized guidelines and standards for the V&V of AI/ML systems; AI-specific certification frameworks; and a focus on human factors, particularly in systems where human-AI interactions are critical. Furthermore, the report called for ongoing research into key areas such as uncertainty management, explainability, and real-time monitoring, which are crucial for the successful deployment of AI/ML in safety-critical aviation systems.

In conclusion, AI/ML technologies hold significant potential for advancing autonomous capabilities in aviation. However, their integration into safety-critical systems requires rigorous safety assurance and certification processes. While traditional V&V techniques provide a foundation, substantial advancements in assurance frameworks, standards, and safety methodologies are necessary to fully realize the potential of AI/ML in the aviation industry.

1 Introduction

An initial study of the use of artificial intelligence (AI) and machine learning (ML) in aircraft can be found more than two decades ago (L. Harison, P. Saunders and J. Janowitz, 1994). But in the last ten years, AI and ML have rapidly progressed with outstanding achievements in image recognition (A. Krizhevsky, 2012), natural language processing (T. Brown, 2020), gaming (D. Silver, 2016), etc. One of the main application areas of AI and ML is unmanned systems, including self-driving cars and unmanned aerial vehicles (UAVs), promoted by DARPA's Grand Challenge (The Grand Challenge, 2004) and Urban Challenge (Urban Challenge, 2007). AI and ML were initially adopted for perception, planning and control in unmanned systems, where AI/ML partially or fully replaces human drivers or pilots in order to improve situational awareness, ensure safer operations, and reduce human drivers or pilots' workload. The fast development of AI/ML technologies and their testing and verification technologies in recent years has led to regulatory body interest, such as by the Federal Aviation Administration (FAA) (C. Wilkinson, J. Lynch, R. Bharadwaj and K. Woodham, 2016) and the European union Aviation Safety Agency (EASA) (EASA AI Task Force and Daedalean AG, 2020).

ML implies a shift to a new software design paradigm, i.e., a data-driven learning paradigm. Combined with AI, it is able to address unseen or unexpected information or situation with its capability of generalization. However, major challenges have emerged, such as AI/ML system understanding, formalization, specification, verification and validation (V&V), and certification. Since most conventional assurance methods used in safety-critical aviation software applications address system behavior rather than data, they do not fit well in the new data-driven learning paradigm. For instance, traditional testing and simulation methods are found limited to address AI/ML correctness, robustness, and safety concerns (AFE 87 - Machine Learning, 2020; SAE G-34 / EUROCAE WG-114, 2021). Moreover, AI/ML also brings other major concerns of predictability, explainability and verifiability required by the design of safety-critical airborne systems. Lack of practice and expertise in AI/ML assurance technologies, standards, and guidance hinders further integration of AI/ML in safety-critical airborne systems.

Our Boeing AI tech team is an active participant in the development of AI/ML technologies for safety-critical airborne systems onboard small and large UAVs in both commercial and military domains. This team also plays an important role in promoting AI/ML assurance technologies and AI/ML standardization efforts. Many members of this team participate in the DARPA Assured Autonomy program, National Aeronautics and Space Administration (NASA)-Boeing Autonomy V&V project, NASA Autonomy V&V Roadmap and Vision 2045 project, and Boeing V&V for

AI-enabled Autonomy project. Supported by internal and external projects, Boeing's team has been exploring AI/ML development, simulation-based and real UAV platform development, platform integration with AI/ML technologies, and evaluation of these technologies. Some preliminary experience and results have been published or discussed in (S. Beland, I. Chang, A. Chen, M. Moser, J. Paunicka, D. Stuart, J. Vian, C. Westover, H. Yu, 2020; Y. He, H. Yu, G. Brat, M. Davies, 2021). In addition, technical exchanges with Boeing's experts from safety, certification, and regulatory affairs have been coordinated in order to better understand and identify the gaps between AI/ML assurance and safety certification.

1.1 AI/ML techniques in autonomous systems

This section presents a survey of existing techniques related to AI/ML, covering typical autonomous functions, AI/ML algorithms, AI/ML assurance techniques, and relevant standardization efforts.

1.1.1 Autonomous functions

Autonomous functions are not a focus of this project, but they help to understand the running context and usage of AI/ML algorithms. The functions presented in this report are mainly related to safety-critical airborne systems. These onboard functions are key functions to achieve autonomous missions with or without human pilots' intervention. The following autonomous functions are the popular ones found in UAV autonomous operations:

- Perception and information fusion
- Mapping and localization
- Planning and decision making
- Flight control
- Coordination and teaming
- Contingency planning

Due to the different nature of these functions, different development processes, data collection, AI/ML algorithm selection, assurance techniques, evaluation metrics, and execution platforms are considered in the AI/ML development and deployment.

1.1.2 Typical AI/ML algorithms

Based on a survey of AI research and development in transportation industries, a short list in Table 1 is given here to show typical AI/ML algorithms used in basic autonomous functions, such as perception, localization, planning, and control. These algorithms are used in many applications domains, such as autonomous vehicles (Y. Ma, 2020) and UAVs (A. Carrio, 2017).

Table 1. Typical AI/ML algorithms used in autonomous systems

Autonomous functions	**AI/ML algorithms**	
Perception	Camera: CNN (+ bounding box regression)	▪ Single shot detectors (YOLO (J. Redmon, 2016), SSD (W. Liu, 2016)) ▪ Focal loss (RetinaNet (T. Lin P. G., 2017) ▪ Region based (FPN (T. Lin P. D., 2017), Faster R-CNN (S. Ren, 2015)) ▪ Atrous convolutions (Deeplab-v3 (L. Chen, 2017))
	LIDAR	▪ 3D CNN
	Radar	▪ DBN trained with Spectral Correlation Function (SCF)
	Acoustic	▪ Partially Shared Deep Neural Network (PS-DNN)
Localization	▪ Graph-based SLAM (Cartographer) ▪ LSTM-RNN ▪ Particle filters (Gmapping) ▪ Visual odometry (SVO, DSO, ORB-SLAM (R. Mur-Artal, 2015))	
Planning	▪ A* ▪ D* ▪ Rapidly-exploring random tree (RRT)	

Autonomous functions	AI/ML algorithms
	▪ Probabilistic roadmap (PRM) ▪ Fast Marching Trees (FMT*) ▪ Stable Sparse Trees (SST*) ▪ Inevitable Collision State Avoidance (ICS-AVOID)
Control	▪ Model identification adaptive controllers (MIACs) ▪ Model reference adaptive controllers (MRACs) ▪ Value function approximation (Deep Q learning) ▪ Imitation learning (Inverse RL, Dagger)

Even though this is a small list, we still observe a good number of AI/ML algorithms. These algorithms are very different in nature. Their V&V techniques are required to consider not only the characteristics of the AI/ML algorithm, but also the requirements, training and testing data, and ML compiling and implementation. These considerations already exist in the V&V of conventional software, but extensions and modifications are still needed to address new features of AI/ML.

1.1.3 AI standardization efforts

In addition to the exploration of assurance technologies, AI development processes, guidance, standards and certification also attract considerable efforts made by SAE G-34/EUROCAE WG114 AI in Aviation (SAE G-34 / EUROCAE WG-114, 2021), ANSI/Underwriters Laboratories (UL Standards & Engagement, 2019), National Institute of Standards and Technology (NIST) (Phillips, Hahn, Fontana, Broniatowski, & Przybocki, 2020), ASTM International (F38.01, 2017), ANSI UASSC Working Group (2020), etc. Most of the standards produced by these organizations will be considered in this project when they are published and ready for the aerospace industry.

1.2 Tasks

This project mainly explores major challenges, potential solutions, technology and process assessment related to AI/ML assurance techniques, AI/ML implementation techniques, AI safety

risk mitigation approaches, and future AI guidance/regulations. Correspondingly, we have the following tasks, as shown in Figure 1: 1) assess current AI/ML assurance techniques; 2) assess current AI/ML implementation techniques; 3) explore safety risk mitigation approaches, and 4) make recommendations for future guidance, standards, and certification.

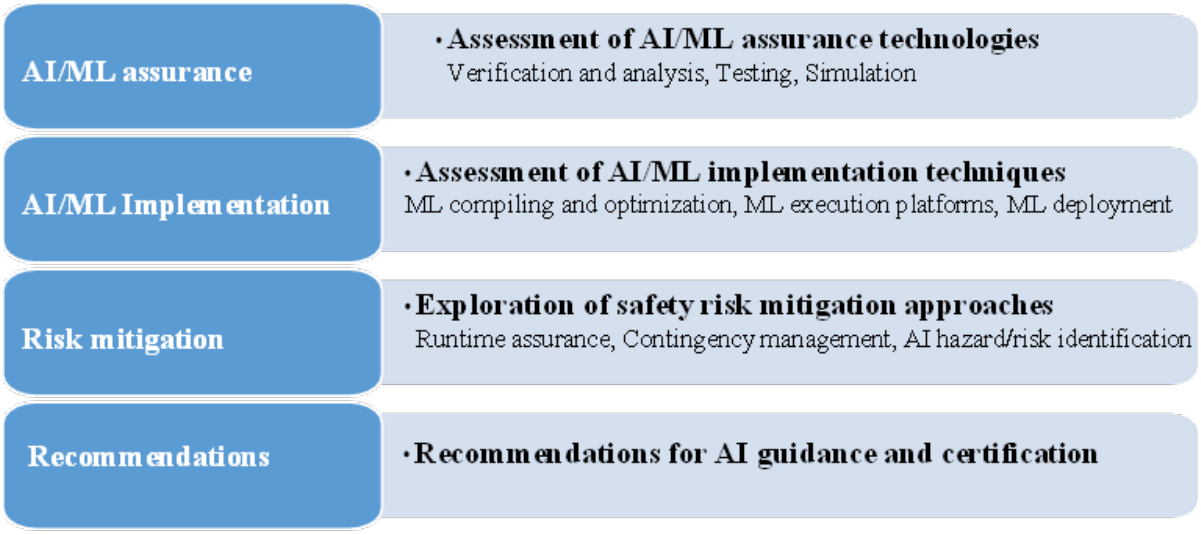

Figure 1. Main technical areas and tasks to achieve project objectives

2 Assessment of AI/ ML V&V, assurance, and implementation technologies

The main objectives of the assessment include a better understanding of the capabilities of the assurance techniques, the mathematical methods to implement the assurance techniques, and the barriers of their integration into current processes and platforms. Based on the research findings, improvement of current assurance techniques or creation of new assurance approaches, techniques, or tools can be planned. This section gives more details of our approach for the assessment of AI/ML V&V techniques and other relevant findings.

2.1 Assessment approach

Our approach has five main activities or steps, shown in Figure 2:

1. A brief survey of AI/ML techniques used in autonomous systems.
2. Identification of technical gaps from public reports.
3. A brief survey of AI/ML assurance techniques. This step summarizes current verification and testing techniques from both research and industry.
4. Assessment of assurance techniques. This step deals with the metrics, properties and potential case studies to assess the assurance techniques.
5. Assessment of integration of the assurance techniques into current process and platforms.

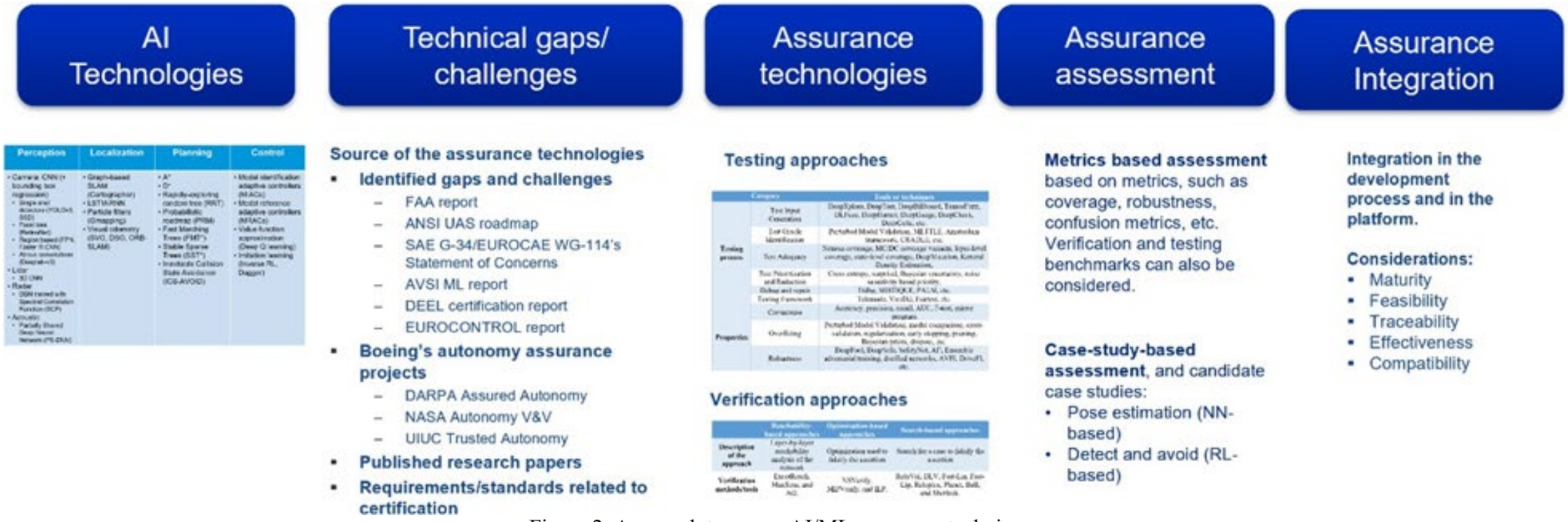

Figure 2. Approach to assess AI/ML assurance techniques

2.1.1 Resources for technical gaps

The survey of technical gaps has been mainly based on identified gaps and challenges; experience from our autonomy projects; published papers and reports; and requirements/standards related to certification:

- **Identified gaps and challenges.** These gaps and challenges are mainly summarized from published reports, such as FAA reports, ANSI UAS roadmap, SAE G- 34/EUROCAE WG-114's Statement of Concerns, AVSI report, DEEL report, and EUROCONTROL report. Newly published reports or other useful sources will also be added during the execution of this project.
- **Boeing's autonomy assurance projects.** These projects include DARPA Assured Autonomy (DARPA, 2019) and NASA-Boeing Autonomy Verification and Validation (Brat, 2019), and the new NASA Autonomy V&V Roadmap project (NASA, 2021).
- **Criteria in published research papers.** A study of shortlisted design-time and operation-time assurance in published in research papers, such as those in (Liu, et al., Algorithms for Verifying Deep Neural Networks, 2020) (J. Zhang, 2020), will also be carried out.
- **Requirements/standards related to certification.** This research will explore certification related requirements defined in existing assurance standards, such as SAE ARP 4754A, SAE ARP 4761, DO-178C, DO-278A, DO-254, DO-330, and DO-333.

2.1.2 Assurance techniques

The assurance techniques considered in this project include testing, verification/analysis and simulation.

2.1.2.1 Artificial intelligence/machine learning testing tools

Statistical testing has been the primary method to verify ML algorithms according to requirements, such as accuracy, distance, and capability of generalization. But with the popularity of ML used in a large number of applications, other research communities have also proposed testing methods to address ML properties similar to those of conventional software, such as correctness and robustness. A comprehensive survey of machine learning testing was performed (J. Zhang, 2020), and Table 2 shows a summary of the surveyed testing tools or techniques.

Table 2. A summary of machine learning testing methods, extracted from (J. Zhang, 2020)

	Category	Tools or techniques
Testing process	Test Input Generation	DeepXplore, DeepTest, DeepBillboard, TensorFuzz, DLFuzz, DeepHunter, DeepGauge, DeepCheck, DeepColic, etc.
	Test Oracle Identification	Perturbed Model Validation, METTLE, Amsterdam framework, CRADLE, etc.
	Test Adequacy	Neuron coverage, MC/DC coverage variants, layer-level coverage, state-level coverage, DeepMutation, Kernel Density Estimation,
	Test Prioritization and Reduction	Cross entropy, surprisal, Bayesian uncertainty, noise sensitivity-based priority,
	Debug and repair	Tfdbg, MISTIQUE, PALM, etc.
	Testing framework	Telemade, VerifAI, Fairtest, etc.
Properties	Correctness	Accuracy, precision, recall, AUC, F-test, mirror program
	Overfitting	Perturbed Model Validation, model comparison, cross-validation, regularization, early stopping, pruning, Bayesian priors, dropout, etc.
	Robustness	DeepFool, DeepSafe, SafetyNet, AI^2, Ensemble adversarial training, distilled networks, AVFI, DriveFI, etc.

2.1.2.2 Artificial intelligence/machine learning verification tools

Due to the weakness of testing in coverage, completeness, robustness, etc., the formal methods community has become very active in developing formal methods-based verification techniques since 2017, particularly inspired by several initial research in this area (Corsi, Marchesini, & Farinelli, 2021; Genin, et al., 2021). A summary of the most recent neural network verification algorithms is summarized by Liu et al. (2020). Table 3 is a summary of identified verification methods and representative tools (Liu, et al., Algorithms for Verifying Deep Neural Networks, 2020).

Table 3. A summary of machine learning verification methods

	Reachability- based approaches	**Optimization-based approaches**	**Search-based approaches**
Description of the approach	Layer-by-layer reachability analysis of the network	Optimization used to falsify the assertion	Search for a case to falsify the assertion
Verification methods/tools	ExactReach, MaxSens, and Ai2.	NSVerify, MIPVerify, and ILP.	ReluVal, DLV, Fast-Lin, Fast-Lip, Reluplex, Planet, BaB, and Sherlock.

2.1.2.3 Artificial intelligence/machine learning simulation tools

Simulation plays a critical role in the development of AI/ML and its assurance technologies. It is a less expensive solution compared to real-world testing and verification. Various simulation tools, such as X-Plane 11 Flight Simulator (Laminar Research, 2021), AirSim (Microsoft, 2021), CLARA (Computer Vision Center and The Embodied AI Foundation, 2021), and Gazebo (Open Source Robotics Foundation, 2021) are used in Boeing projects for various autonomous applications according to their capabilities. X-Plane was the main simulator in this project to show the assessment of V&V techniques.

2.1.3 Assurance assessment

Two methods of assessment are adopted: comparison of selected assurance technologies and evaluation of the selected assurance technologies against simulated or real UAV platforms. Selected assurance technologies are compared based on well-established evaluation metrics, such as coverage, confusion metrics, and assurance metrics. These assessments consider different activities and phases in the AI/ML development life cycle, e.g., requirements, software development, testing and simulation, and platform integration.

On the other hand, selected assurance technologies are evaluated against simulated or real UAV platforms based on case studies that contain AI-enabled autonomous functions. These studies show the strength, weakness, and effectiveness of AI/ML assurance techniques. This case study-based assessment will mainly other studies from Boeing's ongoing projects related to AI/ML V&V.

2.1.3.1 Artificial intelligence/machine learning V&V properties and metrics

V&V properties are mainly derived from AI requirements, particularly from requirement

formalization and analysis activities. Compared with V&V tools, V&V properties are subject to much less change in the long term, as seen in the evolution of V&V techniques in the last decades. These properties are agnostic from the implementation of V&V approaches and tools. We will first report typical examples of V&V properties related to verification/analysis, testing, and simulation. Here are some typical examples of V&V techniques:

- ML verification and analysis: formal-methods-based, empirical, ML-based, statistics-based, etc.
- Testing: formal-methods-based, empirical, ML-based, statistics-based, etc.
- Simulation: flight simulators, game engines, other dedicated sims, hardware in the loop, etc.

2.1.3.1.1 Verification and validation properties related to verification and analysis

Typical examples of properties related to verification and analysis are shown in Table 4.

Table 4. Sample set of verification properties for ML

Examples of V&V techniques (not a full list)	General properties of the requirements/AI component/data	Expected properties that can be checked and improved by using V&V	Additional properties or evaluation metrics	System-level properties	Expected properties of V&V/safety tools
▪ Model checking ▪ Runtime verification ▪ Automated theorem proving ▪ Statistical analysis	▪ Complexity ▪ Verifiability/testability ▪ Fairness Criticality ▪ Modularity ▪ Compositionality ▪ Reusability ▪ Transferability	▪ Accuracy ▪ Correctness ▪ Robustness (AI/ML) ▪ Explainability ▪ Interpretability ▪ Appropriateness of abstraction (verification or analysis model regarding the system) ▪ Model validation, ▪ Stability	▪ Coverage ▪ Confusion matrix and derivations (used in analysis and testing) ▪ Uncertainty	▪ Robustness (of the AI-enabled system, regarding noise, disturbance, etc.) ▪ Reliability	▪ Scalability ▪ Applicability ▪ Usability ▪ Interoperability ▪ Performance ▪ Robustness (of the verification or analysis tool)

2.1.3.1.2 Verification and validation properties related to testing

Examples of testing techniques are automated test generation, regression testing, ML-based testing, stress testing, compliance testing, field testing, statistics-based testing. Typical examples of testing properties are shown in Table 5.

Table 5. Sample set of properties related to testing

General properties of the requirements/AI component/data	Expected properties that can be checked and improved by using V&V	Additional properties or evaluation metrics	System-level properties	Expected properties of V&V/safety tools
▪ Complexity ▪ Verifiability/testability ▪ Fairness ▪ Criticality ▪ Modularity ▪ Compositionality ▪ Reusability ▪ Transferability	▪ Accuracy ▪ Correctness ▪ Robustness ▪ Explainability ▪ Interpretability ▪ Fairness ▪ Appropriateness of abstraction (test model regarding the system) ▪ Model validation ▪ Stability	▪ Confusion matrix and derivations (used in analysis and testing) ▪ Generalization gap ▪ Performance ▪ Coverage ▪ Uncertainty	▪ Safety region/boundary accuracy and coverage ▪ Robustness (of the AI-enabled system, regarding noise, disturbance, etc.) ▪ Reliability ▪ Mileage/flight hours	▪ Scalability ▪ Usability ▪ Interoperability ▪ Statistical properties (test data, scenarios, process, etc.) ▪ Applicability ▪ Performance ▪ Adequacy ▪ Effectiveness ▪ Robustness (of the testing tool)

2.1.3.2 Boeing AI/ML assurance projects

Boeing's most recent assurance projects focus on development and evaluation of AI/ML assurance techniques dedicated to autonomous systems based on real-world challenges and case studies, e.g., DARPA Assured Autonomy, NASA-Boeing Autonomy V&V, and Boeing V&V for AI-enabled Autonomy. New assurance technologies under development address both design-time and operation-time assurance, considering simulation-based and real UAV platforms' characteristics and constraints during technology development, integration, and evaluation.

In the air domain of DARPA Assured Autonomy, five research teams have been working on design-time verification, simulation, testing, runtime monitors, and dynamic assurance to address autonomy challenges proposed by the Boeing team. As the real platform provider, the Boeing team performs platform-based integration and technology evaluation. Most assurance technologies are developed based on formal methods, using methods of reachability, over-approximation, and optimization and search. The ML components to be assured in this project, most of which are neural networks-based, feature reinforcement learning and supervised learning. The Boeing team develops technical requirements, ML components, real and simulation platforms, and assurance technology evaluation. This collaboration between research and industry makes it possible to identify the real gaps between assurance technologies and autonomous platforms.

In the NASA-Boeing Autonomy V&V project, two use cases are used to explore the challenges of verification of AI-enabled autonomy: Autonomous Centerline Tracking (ACT) and Object Detection & Classification (ODC). Four V&V techniques have been evaluated, improved, and demonstrated, including neural network verification and analysis (Prophecy (D. Gopinath, 2019)), advanced system analysis and testing (SYSAI), runtime monitors (R2U2 (J. Schumann, 2015)), and safety assessment (fmdTools). The Boeing team provides machine learning models, training and testing data, simulation environment, requirements and V&V objectives, and integration environment. NASA researchers add new features and functions to their original V&V techniques to address Boeing's new machine learning V&V needs. The final demonstration shows how these newly developed features and functions achieve the expected V&V objectives. Potential use of these V&V techniques in certification is also being explored.

2.2 Safety assurance in reinforcement learning

Reinforcement Learning (RL) is a method in which an agent learns to make decisions through trial-and-error interactions with its environment, guided by a reward system as shown in Figure 3. With improvement in environment modeling and computational power, RL methods provide

promising solutions to problems of sequential decision making in the aerospace domain. This enablement has led to the use of RL in lot of aerospace applications (Razzaghi, et al., 2022) such as collision avoidance, air traffic flow management, aircraft controls, flight planning, and many more.

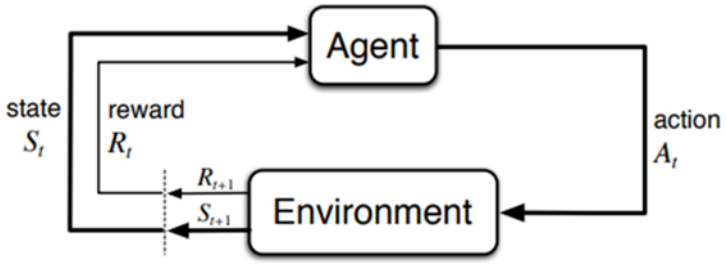

Figure 3. RL Agent interaction with the environment

In this first part of the report, the key concepts in defining RL problems are introduced. Then, some of the important algorithms in Deep RL are presented, as there is increased use of neural network-based function approximators in aviation. Finally, a systematic review of safe implementation of RL algorithms is provided.

2.2.1 Core concepts in RL

Key components in RL problems include agents, environment, states, actions, and rewards. In RL, the agent's primary objective is to maximize cumulative rewards over time. This process is mathematically represented using the concept of Markov Decision Process (MDP) (Puterman, 2014). A MDP is defined as a tuple of $\langle S, A, P(s'|s,a), R(s,a), \gamma \rangle$. The tuple elements are defined as follows:

- Set of states S, (continuous or discrete): At every decision epoch the agent occupies a particular state. These states can be modeled as continuous or discrete.
- Set of actions A , (continuous or discrete): From each state the agent has access to a set of actions which can be executed. These actions can also be modelled as any combination of continuous or discrete and deterministic or stochastic.
- Transition function $P(s'|s,a)$: The state of the agent in the next decision epoch is determined by the state transition function, which defines the probability distribution of agent's next state.
- Reward function $R(s,a,s')$: The agent receives an immediate reward when in a particular state and executing an action.

- Discount factor γ: The value of reward received by the agent over the time horizon of the problem is discounted by this factor. Discount factor ranges from $0 \leq \gamma \leq 1$. $\gamma = 0$ signifies agent-only values immediate reward and $\gamma = 1$ signifies agent values immediate and future rewards equally for the horizon of the problem. However, for convergence in infinite horizon problems, $\gamma < 1$ is considered.

Central concepts in RL are value functions—including both the state-value $V(s)$ and action-value $Q(s, a)$—and policies. Value functions estimate the expected cumulative future reward from a given state or state-action pair under a particular policy. This helps in estimating how beneficial it is for the agent to be in a specific state. A policy is defined as a mapping from a particular state to the action probability.

The value function of a state s under policy π is defined as follows:

$$v_\pi(s) := \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right], \text{ for all } s \in S$$

Similarly, we can define the state-action value function for policy π as follows:

$$q_\pi(s, a) := \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a \right]$$

Generally, solving RL problems involves determining the optimal policies (π_*) that provide large rewards over the horizon of the problem, sharing the same optimal state value function $v_*(s)$ or action-value function $q_*(s, a)$. Dynamic programming provides methods to determine the optimal policies. Techniques like value iteration (VI) and policy iteration (PI) fall under this category. VI determines the optimal value function, and optimal policy is inferred from it. On the other hand, PI directly computes the optimal policy and sets the basis for developing very powerful policy gradient-based methods, as discussed in the following section.

The assumption of perfect world in presence of imperfection and high computational expense of using either PI and VI motivates the use of techniques based on Monte-Carlo (MC) methods and temporal difference (TD) learning (Sutton & Barto, 2018). However, MC and TD methods require maintaining explicit values of state in a tabular format, which becomes impractical when dealing with problems of arbitrarily large state spaces. For additional details on tabular format, readers are directed to (Sutton & Barto, 2018). In such cases, determining optimal values or policies is impractical. Instead, approximate solutions are obtained using limited computational resources.

Some of the other key concepts in RL are off-policy and on-policy learning; model-based and model-free methods; and exploration versus exploitation.
In off-policy learning, the agent learns to estimate and evaluate a policy using experience gathered from following a different policy. An example is Q-Learning. In on-policy learning, the agent learns to estimate and evaluate a policy from experience obtained following that policy. An example is on-policy MC control (Sutton & Barto, 2018).

A model-based method makes use of the transition dynamics of an environment, whether learned or already known. Model-free methods do not consider the state-transition dynamics. The core of both model-based and model-free methods is the computation of value function.

Exploration versus exploitation describes the trade-off between exploring new actions to discover their potential rewards and exploiting known strategies to maximize current gains. A technique to manage the exploration versus exploitation balance is using ϵ −greedy policy with reference to a state-action value $Q(s,a)$, that is:

$$\pi(a|s) = \begin{cases} \arg\max_a Q(s,a) \ \ with \ \ prob \ 1-\epsilon+\dfrac{\epsilon}{|A|} \\ a' \neq \arg\max_a Q(s,a) \ with \ prob \dfrac{\epsilon}{|A|} \end{cases}$$

2.2.2 Deep RL

Traditional RL methods struggle with large state spaces or continuous actions, making tabular methods intractable. This section introduces function approximation in RL, where neural networks estimate value functions or policies, making the problem tractable. Utilization of deep neural networks representing the RL agent enables the handling of high-dimensional state spaces such as an image-based inputs.

2.2.2.1 Popular deep reinforcement learning algorithms

Deep reinforcement learning (DRL) merges RL principles with deep neural networks. Key methodologies include Deep Q-Networks (DQN), policy gradient methods, and actor-critic methods. DQN uses deep networks for Q-value approximation, minimizing loss functions based on the Bellman equation (Sutton & Barto, 2018). Policy gradient methods optimize a parameterized policy $\pi_\theta(a|s)$ directly, which is beneficial in continuous action spaces. Actor-critic methods use two models: an actor proposing actions and a critic evaluating them. The mathematical relationship between actor and critic stabilizes and improves learning. Figure 4 shows a non-exhaustive list of some of the popular Deep RL algorithms. For additional list of RL algorithms readers are directed to (OpenAI, 2018).

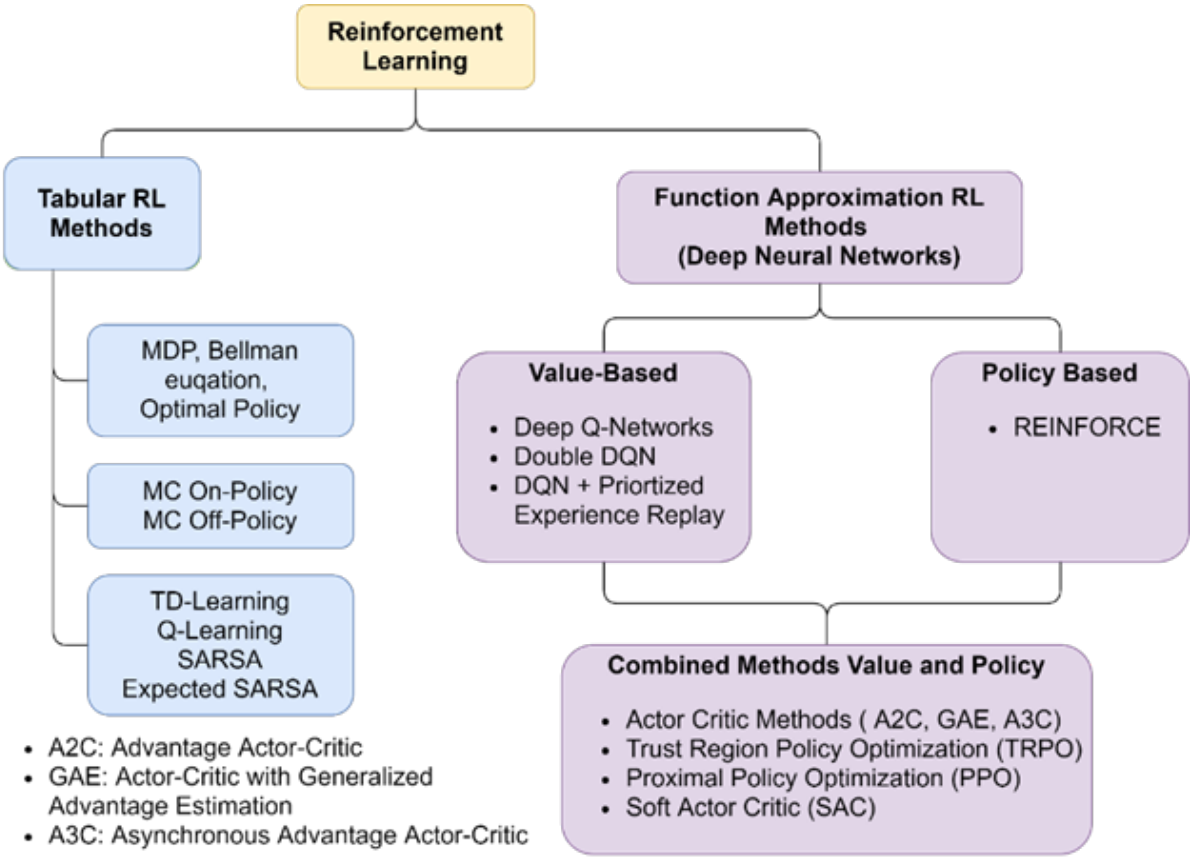

Figure 4. A non-exhaustive taxonomy of RL algorithms

2.2.2.1.1 Deep Q-Network

DQN, a landmark in DRL, integrates deep neural networks with the Q-learning algorithm (Mnih, et al., 2013). It approximates the Q-value function using deep learning, enabling the agent to learn optimal policies in high-dimensional spaces. The key innovation in DQN is the use of experience replay, where the agent stores past experiences and samples them randomly to break the correlation in observation sequences. This process stabilizes learning. Another critical aspect is the use of a target network, a periodically updated clone of the Q-network, which helps in stabilizing the training process. The DQN algorithm from Mnih, et al. (2013) with experience replay is shown in Figure 5:

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Figure 5. DQN algorithm with experience replay

2.2.2.1.2 Policy gradient methods

Policy gradient methods directly optimize the policy function $\pi_\theta(a|s)$, where θ represents the parameters of the neural network. One of the core advantages of policy gradient methods is their effectiveness in continuous action spaces. Algorithms like REINFORCE (Sutton & Barto, 2018) utilize the policy gradient theorem, which provides a gradient estimation for policy improvement. Advanced variants, such as Trust Region Policy Optimization (TRPO) (Schulman, Levine, Moritz, Jordian, & Abbeel, 2017) and Proximal Policy Optimization (PPO) (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017), improve training stability and efficiency by ensuring small updates to the policy.

PPO has been widely adopted in various applications of RL due to its effectiveness and balance between complexity and performance. Central to PPO is its objective function, which involves a clipping mechanism to control policy updates. This function uses the ratio $r(\theta) = \frac{\pi_\theta}{\pi_{\theta_{old}}}$, comparing the new policy π_θ to the old $\pi_{\theta_{old}}$, where θ denotes policy parameters. The objective function is $min(r(\theta)A_t, clip(r(\theta), 1-\epsilon, 1+\epsilon)A_t)$, with A_t as the advantage function, indicating the relative value of actions, and ϵ as a hyperparameter setting the clipping range. This clipping restricts the update magnitude, preventing drastic policy changes and ensuring smoother learning. Additionally, PPO alternates between sampling data via interaction with the environment and optimizing the clipped objective, using multiple epochs of stochastic gradient ascent per data sample. This process contrasts with TRPO, offering similar benefits but with

simpler implementation and fewer computational demands. The PPO-clip algorithm from OpenAI (2018) is shown in Figure 6:

Algorithm 1 PPO-Clip

Input: initial policy parameters θ_0, initial value function parameters ϕ_0
for $k = 0, 1, 2, \ldots$ **do**
  Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
  Compute rewards-to-go $\hat{R}_t$.
  Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
  Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t),\ \ g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

  typically via stochastic gradient ascent with Adam.
  Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

  typically via some gradient descent algorithm.
end for

Figure 6. PPO-clip algorithm

2.2.2.1.3 Actor-critic methods

Combining value-based and policy-based approaches, actor-critic methods use two networks: an actor that suggests actions and a critic that evaluates them. Algorithms like Asynchronous Advantage Actor-Critic (A3C) (Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, Koray Kavukcuoglu, 2016) and Soft Actor-Critic (SAC) (Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, Sergey Levine, 2018) have demonstrated success in various environments. A3C, for instance, introduces asynchronous updates to improve learning speed and stability, while SAC focuses on maximizing a trade-off between entropy (exploration) and reward, enhancing the exploration capabilities of the agent.

2.2.3 Review of safety assurance in RL

The previous section discussions were centered around the core concepts and review of various Deep RL algorithms. While RL holds great promise for optimizing outcomes and solving intricate problems, its deployment in safety-critical applications raises some challenges and concerns.

Safety verification of RL agents is of paramount importance when it comes to systems that directly impact human lives, infrastructure, or the environment (Shangding Gua, Long Yang, Yali Du, Guang Chen, Florian Walter, Jun Wang, Yaodong Yang, Alois Knoll, 2023; Gu, n.d.; Javier Garcia, Fernando Fernandez, 2015). In safety-critical applications like aerospace, verifying the desired and undesired behavior of RL agents is of utmost importance for certification. This report delves into the crucial topic of verifying safety in reinforcement learning, focusing on application in safety-critical contexts.

This report aims to provide a comprehensive overview of the challenges, methodologies, and best practices involved in incorporating safety verification measures into RL algorithms. We will explore the intricacies of defining safety, mitigating risks, and developing robust RL systems that can operate reliably under various conditions.

2.2.3.1 Methodology

The main goal of this section is to identify, review and summarize the state-of-the-art techniques used for safety verification in RL implementations. The topic of safety verification in RL is vast. To limit the scope of our review, we iteratively down-selected the following keywords used in our search study:

{ reinforcement learning OR deep reinforcement learning } AND
{ safety critical OR safety assurance OR certification OR verification and validation }
OR
{ automotive OR aerospace OR driving }
OR
{ collision avoidance OR conflict resolution OR air traffic deconfliction }

As shown in our keywords (Figure 7) the search was not limited to only aerospace. Automotive was also included, as lot of research on safety verification of RL is conducted. Using the keywords, papers were searched in the following databases: Google Scholar, the American Institute of Aeronautics and Astronautics (AIAA), Neurips, the Institute of Electrical and Electronics Engineers (IEEE), , Proceedings of Machine Learning Research, Proceedings of Association for the Advancement of Artificial Intelligence (AAAI) Conference on AI, arXiv, NASA Technical Reports Server (NTRS), Formal Methods Conference, and the International Conference on Machine Learning (ICML). To review state-of-the-art techniques, the publication date was limited from Jan 2015 till September 2023. Figure 7 shows the number of papers selected in each year from 2015 to 2023, totaling to 72 papers.

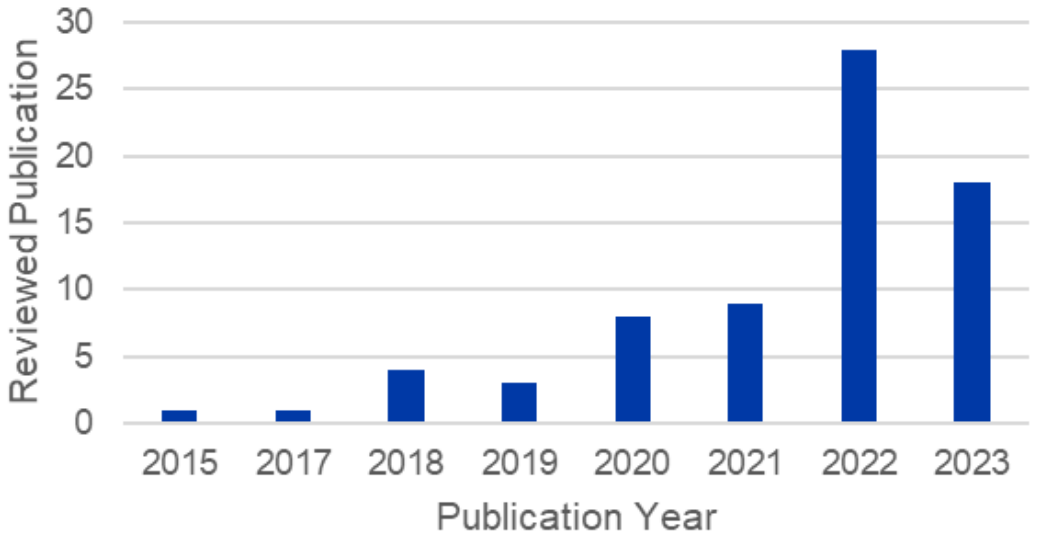

Figure 7. Number of papers studied on safety verification in reinforcement learning implementations

Figure 8 shows the affiliation and venues of the selected papers. The affiliation information was obtained from screening the paper authors and their affiliation. It was observed that the safety verification of RL was well investigated by the industry, as 25% of the papers were either industry alone or collaboration between the industry and academia.

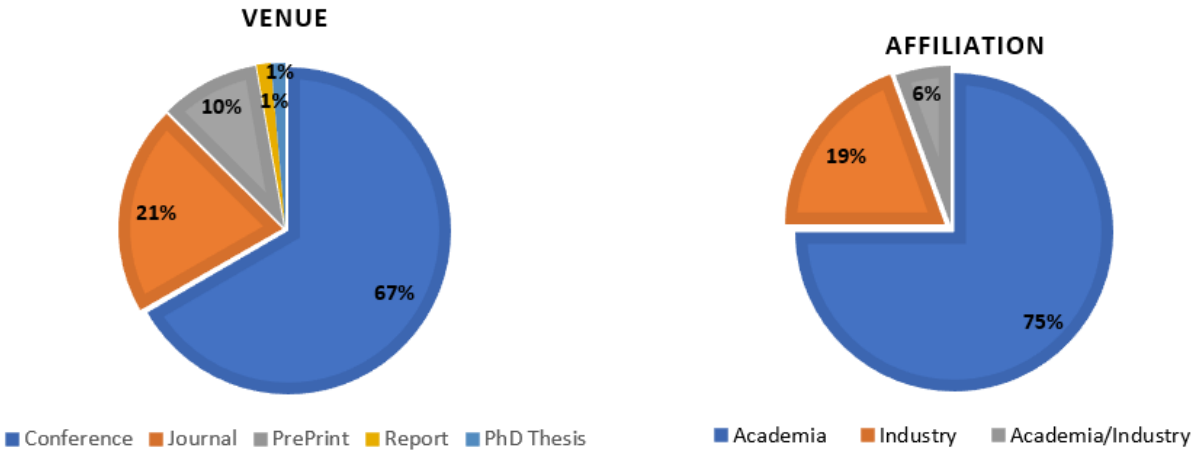

Figure 8. Venue and affiliation of paper reviewed

2.2.3.2 Review

In this section, a summary of the state-of-the art methods for the safety verification of RL implementations are presented. The summary is loosely categorized as follows:

- Verifying behavior of the RL agent
- State space robustness
- Reward modeling
- Stability in RL based controllers
- Out-of-distributions considerations
- Testing RL agent.

2.2.3.2.1 Verifying behavior of the RL agent

Understanding why the RL agent takes a particular action is important after training to verify model behavior. Guo & Wei (2022) and Guo, Zhou, & Wei (2022) convert complex Deep RL models into a shallow soft decision trees, which are used to provide a visual explanation of agents' behavior in each step. This approach is applied to provide assurance for high-dimensional aircraft separation problems. A similar approach is used to visualize the internal states and facilitate coherent explanation of agents' decisions using object saliency maps, as proposed by Iyer, et al. (2018). Object saliency maps and screenshots have proven equally helpful to humans in predicting a deep reinforcement learning network's (DRLN's) behaviors. Iyer, et al. have evaluated approach via computational and human experiments on simple Ms. Pacman game play, although efficacy on real-world problems has not been evaluated. Dai, Ouyang, Zheng, Long, & Duan (2022) claim that saliency maps-based techniques cannot explicitly present the cause and effect between agents actions and observations. Their proposed method focuses on interpreting the behavior of Deep RL agents by analyzing inference procedures. It introduces a conceptual embedding technique to regulate latent representation spaces of deep models, producing interpretable causal factors aligned with human concepts.

Formal methods-based tools have gain popularity recently in evaluating deep neural networks-based RL agents (Liu, Cofer, & Osipychev, Verifying an Aircraft Collision Avoidance Neural Network with Marabou, 2023; Fulton & Platzer, 2018; Genin, et al., 2021; Bacci & Parker, Verified Probabilistic Policies for Deep Reinforcement Learning, 2022). One such approach is presented by Liu, Cofer, & Osipychev (2023), focusing on the safety verification of neural network-based collision avoidance systems in aircraft using Marabou, a neural network verification tool. This research highlights the complexity and importance of ensuring the

reliability and safety of critical systems. The authors in this paper have verified the minimum separation distance and conducted robustness analysis. Marabou was able to identify multiple scenarios where neural network outputs resulted in collision. These scenarios could be used for retraining the collision avoidance neural network agent. Fulton & Platzer (2018) combine formal methods with verified runtime monitoring for RL agents to provide safe control in dynamic environments. This integration facilitates a balance between learning from data and adhering to formally verified safety constraints.

An approach to guarantee safety in model-free RL algorithm is proposed by Modares, Sadati, Esmaeili, Yaghmaie, & Modares (2023). It introduces a data-driven safety certifier designed to intervene with the actions of an RL agent, ensuring both safety and stability without requiring an explicit system model. This approach is applied in linear parameter-varying systems with unstructured bounded uncertainties or disturbances, common in aerospace and robotics systems. The proposed approach is limited to convex safe sets. A model-based approach to certify the system properties, such as stability, safety etc., is described by Wang, et al. (2023) for Cyber Physical Systems. It proposes a framework that jointly conducts RL and formal verification, solving a novel bilevel optimization problem. This method significantly improves the ability to find feasible controllers with certificates ensuring system safety and stability. Similar to the previous two approaches providing guarantees of safety of RL agent (Jin & Lavaei, 2020), this method provides a general framework certifying stability of RL for nonlinear systems and applies it to multi-agent flight formation and power system regulation problems. Another approach to certifying safe learning is proposed by Luo & Tengyu (2021) called Co-trained Barrier Certificate for Safe RL (CRABS). The barrier certificates are obtained via adversarial training and ensure the policy's safety assuming calibrated learned dynamics.

Reachability analysis is also used to determine the safety of the RL agent during the learning process. Reachability analysis is a formal method used in systems engineering and computer science to determine the set of all possible states a system can reach when starting from a specific initial configuration, given a set of inputs and constraints. Hsu, Rubies-Royo, Tomlin & Fisac (2021) introduce the Discounted Reach-Avoid Bellman Equation (DRABE) method for solving reach-avoid optimal control problems. It demonstrates the effectiveness of this approach in various scenarios, ranging from simple point particle systems to complex game environments.

Sun, Zheng, Hazzanzadeh, & Liang and (2022) and Seo, Sahoo, & Vamvoudakis (2023) explore the robustness of RL in the face of adversarial threats, which is particularly crucial in aerospace applications where safety and security are paramount. Seo, Sahoo, & Vamvoudakis (2023)

propose a verification process for attack-mitigating control that is grounded in RL. The verification is conducted using VerifAI toolkit.

Sharifi, Yildirim, & Fallah (2023) and Hasanbeig, Abate, & Kroenig (2020) investigate approaches to enhance safety in autonomous driving. In Sharifi, Yildirim, & Fallah (2023), studies integrate symbolic reasoning with deep RL, ensuring logical constraints are respected, which is critical in navigating complex and unpredictable traffic environments. This approach allows learning autonomous driving policy while actively engaging with the real world. Another approach is proposed by Hasanbeig, Abate, & Kroenig (2020) where policies are represented as temporal logic formulas. The algorithm automatically manages the trade-off between safety during the training and exploring the environment to achieve the linear temporal logic objective.

2.2.3.2.2 Robustness to state-space perturbations

Adversarial approach to improve robustness of RL agent is shown by perturbing the input space (Lutjens, Everett, & How, 2019). The proposed approach guaranteed lower bounds on state-action values during execution to identify and choose the optimal action under a worst-case deviation in input space due to possible adversaries or noise. This method is applied to DQN for collision avoidance. However, applying adversarial training methods applied to RL are ineffective, as claimed by Zhang, et al. (2020). The authors propose a state adversarial MDP (SA-MDP), which significantly improved the robustness property of deep RL agents trained using PPO, DQN algorithms.

An interesting approach to robustify the agent against state space noise is proposed by Wu, Sibai, & Vorobeychik (2022), where the observations for RL are predicted from a supervised learning network that is fine-tuned during the during RL. Their algorithm also computes the lower bounds of the adversarial perturbations to the observations providing safety certification for a partially supervised RL (PSRL) agent. Xu, et al. (2022) provide a review of various methods for robust RL against uncertain state observations.

Liang, Sun, Zheng, & Huang (2022) address the vulnerability of deep RL policies to adversarial perturbations. Traditional robust training methods either overlook the robustness of long-term rewards or double the computational load by training both agents and attackers. Liang, Sun, Zheng & Huang propose a novel framework, Worst-case-aware Robust RL (WocaR-RL), which efficiently estimates and optimizes the worst-case reward of a policy under attacks without extra samples for learning an attacker. WocaR-RL claims to outperform existing methods in robustness and efficiency across multiple environments. The framework integrates three key mechanisms: worst-attack value estimation, worst-case-aware policy optimization, and value-

enhanced state regularization, improving long-term robustness without significantly increasing training complexity. Experiments demonstrate that WocaR-RL achieves state-of-the-art performance under various attacks and offers higher training efficiency compared to other methods.

Another way to make RL agents robust to failure scenario is by pruning the state space to represent critical scenarios from naturalistic driving data. The background agents learn what adversarial maneuver to execute through a dense deep-reinforcement-learning (D2RL) approach, in which Markov decision processes are edited by removing non-safety-critical states and reconnecting critical ones so that the information in the training data is densified (Feng, Sun, & Yan, 2023). D2RL enables neural networks to learn from densified information with safety-critical events and achieves tasks that are intractable for traditional deep-reinforcement-learning approaches.

2.2.3.2.3 Safety and stability in RL-based controllers

Kumar, Levine, & Feizi (2021) introduce a novel method called *policy smoothing* aimed at enhancing the robustness of RL models against adversarial attacks. This method incorporates a smoothing technique around the action space, ensuring the policy's stability under small perturbations. Key challenges addressed include maintaining performance while achieving robustness and handling the high-dimensional nature of the action spaces in RL. A similar approach of policy smoothing is proposed in the framework for certifying robustness in RL policies against adversarial attacks (Wu, et al., 2022). It employs local and global smoothing algorithms on Q-functions to ensure action and reward robustness. The paper tackles challenges in estimating Q-function output ranges and demonstrates Certifying Robust Policies' (CROP's) effectiveness through various RL environment experiments. Similar policy smoothing technique is also applied for multi agent reinforcement learning problems (Ronghui Mu, Wenjie Ruan, Leandro Soriano Marcolino, Gaojie Jin, Qiang Ni, 2023).

Approaches based on look-ahead or reachability type analysis have been incorporated while training the RL agent to improve the safety of the agent while learning. Baheri, et al. (2020) have trained a Recurrent Neural Network (RNN) to dynamically learn safety patterns from the surrounding environment for autonomous driving. The RNN and safety heuristics are used with the Double Deep Q-Network (DDQN) agent during training to improve the safety of the agent while learning in a highly changing environment.

Selim, et al. (2022) present the Black-box Reachability-based Safety Layer (BRSL) for safe RL in robotics, which does not require prior knowledge of robot and environment models. BRSL

ensures safety through data-driven reachability analysis and a novel method to adjust reachable sets to avoid collisions. It integrates an online-learned environment model that does not affect safety guarantees. Evaluated in four robot motion planning tasks, BRSL effectively maintained safety while achieving high rewards compared to other methods. A similar approach by Shao, Chen, Kousik, & Vasudevan (2021) presents Reachability-Based Trajectory Safeguard (RTS), a safety layer for RL in continuous control, especially for robots. RTS employs reachability analysis to ensure safety during training and operation, precomputing a Forward Reachable Set (FRS) for the robot. During RL, the agent selects from precomputed trajectories in a receding-horizon manner, with the FRS assessing and adjusting the safety of chosen paths. RTS is validated in simulations with various robot models, outperforming existing methods in safety and task completion. Another approach using reachability analysis by Dong, Zhao, Wang & Huang (2022) is used to verify the safety of the trajectory of agent while learning for a Deep Deterministic Policy Gradient (DDPG) agent. Uncrewed Underwater Vehicle (UUV) docking at the harbor was used as the environment to check the efficacy of the implementation.

An approach to learning safely in real-world environment involves utilization of control barrier functions during RL (Cheng, Orosz, Murray, & Burdick, 2019; Choi, Castaneda, Tomlin, & Sreenath, 2020). Cheng, Orosz, Murray, & Burdick (2019) propose a novel RL-Control Barrier Function (CBF) control synthesis algorithm that guarantees safety of the learning process with high probability, irrespective of the RL algorithm used. They have tested their approach on inverted pendulum and car following examples with TRPO and DDPG agents. However, a shortcoming of this approach is that, since calculating the safe set in a computationally intensive environment, they assume availability of such safe sets used in CBF. If safe sets are not available, their approach might not work. Choi, Castaneda, Tomlin, & Sreenath (2020) present a novel RL framework for safety-critical control in systems with model uncertainty, integrating Control Lyapunov Function (CLF)- and CBF-based quadratic programs (QPs) with RL. The framework, RL-CBF-CLF-QP, learns model uncertainties in dynamic constraints, enhancing safety and stability in uncertain environments. It is validated using simulations of a bipedal robot navigating challenging terrains, demonstrating robustness and effectiveness in adapting to model uncertainties.

Shielding is similar to CBFs. Essentially, this involves adding a safety layer to the DRL algorithm that acts as a shield, preventing the system from taking actions that could lead to unsafe or undesirable outcomes. Harris & Schaub (2020) discuss the use of shielded DRL for safe, autonomous spacecraft command and control, emphasizing performance and safety guarantees.

2.2.3.2.4 Out-of-distribution considerations in RL

Danesh & Fern (2021) addresses the challenge of detecting out-of-distribution (OOD) dynamics in RL. It introduces specialized benchmarks for evaluating OOD detection methods in various simulated environments, reflecting real-world scenarios. The study compares traditional statistical approaches with deep learning-based techniques, assessing their effectiveness in these benchmarks. Results indicate a need for improved OOD detection in RL, highlighting the gap in handling complex, dynamic environments and underscoring the importance of advancing these methods for RL system safety and reliability.

Another useful consideration of OOD detection is presented by Selby & Staley (2022). The paper presents a methodology for determining when to use a DRL policy, particularly in safety-critical systems. It introduces a "decision metric" based on uncertainty estimation techniques, modified Random Network Distillation (RND), and Kalman filtering. This metric helps decide when to replace the DRL policy with another decision-making process, ensuring the system's safety and reliability. The approach is demonstrated using the Meta Arcade environment, with experiments showing its effectiveness in out-of-distribution detection and decision-making.

2.2.3.2.5 Reward modeling

Amodei, et al. (2016) discuss the safety challenges of using AI attributed to unintended and damaging behavior. Challenges presented in this paper include negative side effects, reward hacking, scalable oversight of agent behavior, safe exploration, and robustness to distributional shifts. Of importance to RL is how to model the reward function to avoid negative side effects.

A solution to model rewards relevant to aerospace domain is presented by Cone, Owen, Alvarez, & Brittain (2023). The authors use a surrogate optimizer to model rewards for a DRL agent. Using this surrogate optimizer, they were able to increase safety and operational capability of the agent for collision avoidance.

2.2.3.2.6 Testing and evaluation of RL agents

Following agent training, it is important to test and evaluate DRL algorithms in a real-world environment. Shim, et al. (2023) study real-time optimal route planning for unmanned aerial systems (UAS) and urban air mobility (UAM) using DRL. Focusing on autonomous collision avoidance in dynamic environments, the authors apply DRL methods, specifically DQN, PPO, and Twin Delayed DDPG (TD3), for UAV route generation. Flight tests validate the practicality of these methods, demonstrating their effectiveness in real-world scenarios. A lot of lessons were learned during flight tests; inaccurate flight planning and flight controller instability in certain

scenarios lead to an overshoot-induced crash. This information could be fed back in the DRL training phase to improve performance.

RTA architectures have also been useful in ensuring agent safety in the real world (Cofer, et al., 2022; Mussau, Hamilton, Lopez, Robinette, & Johnson, 2022; Chen, et al., 2022). In these approaches, the system switches from the performance-based DRL controller to a safe controller in certain scenarios. For example, in *Flight Test of a Collision Avoidance Neural with Run-Time Assurance* (Cofer, et al., 2022), the RTA switches from a collision avoidance deep neural network (DNN) to a backup avoidance planner. Chen, et al. (2022) switch from a performance-based DRL controller to a safe velocity-obstacle baseline controller for obstacle avoidance in an autonomous car. A challenge with RTA architecture is identifying of unsafe conditions. Lazarus, Lopez, & Kochenderfer (2020) propose the design of a meta-controller to identify unsafe regions with high accuracy using RL. They successfully applied this method for terminal recovery conditions on a hexacopter.

An approach alternative to flight test for RL is proposed by Tambon, Majdinasab, Nikanjam, Khomh, & Antoniol (2023) and Raz, et al. (2022). It leverages existing taxonomies of faults to create relevant mutation operators for RL and uses a heuristic to generate test cases. Tambon et al. (2023) demonstrates how the design of mutation-killing definitions can significantly affect the detection of mutations and the generation of test cases, enhancing the testing capability in RL systems. This study contributes to understanding and improving the verifiability and reliability of DRL systems. Raz, et al. (2022) have proposed a test and evaluation framework based on Systems Engineering for AI (SE4AI), using Shapley Additive Explanations (SHAP) to examine RL decision-making, and incorporate validation of RL outputs with known and accepted solutions. They have applied this framework for test and evaluation of high-speed aerospace vehicle descent problems.

Hobbs, et al. (2023) present a Systems Theoretic Process Analysis (STPA) of a Neural Network Control System (NNCS) bounded by RTA for aircraft. It focuses on applying Systems Theoretic Accident Models and Processes (STAMP) for identifying safety constraints in a hypothetical flight test.

Chai, et al. (2022) present a hierarchical deep learning control framework for mobile robots. This framework consists of an upper-level motion planning layer and a lower-level waypoint tracking layer. Using a recurrent deep neural network (RDNN) for motion planning and DRL for collision avoidance, the approach significantly improves autonomous exploration, motion planning performance, and training efficiency by learning the policy directly from demonstrations. The

framework is validated through simulation and experimental case studies, demonstrating its superiority in fast trajectory planning and control in uncertain environments.

Braun, Marb, Angelov, Wechner, & Holzapfel (2023) use Q-network to identify input sequences that lead to detrimental system behavior. This approach is applied to identifying failure scenarios for hover flight controls of electric vertical take-off and landing (eVTOL) aircrafts.

2.2.3.3 Summary of DRL verification tools

This section summarizes various tools available for verification of RL, specifically neural networks-based deep RL.

Tools	Description
Property Verifier (ProVe) (Corsi, Marchesini, & Farinelli, 2021)	It is a formal verification tool for trained neural networks, particularly those that encode decision-making tasks for DRL. It is based on interval analysis to verify safety properties in networks with large input spaces.
MOdel SAfe Intelligent Control (MOSAIC) (Bacci & Parker, 2020)	It addresses the verification problem as a finite-horizon analysis of a continuous-space, discrete-time Markov process, employing a combination of abstract interpretation and probabilistic model checking to calculate upper bounds on failure probabilities.
MARABOU (Katz, et al., 2019)	It is an Satisfiability Modulo Theories (SMT)-based tool that can answer queries about a network's properties by transforming the queries into constraint satisfaction problems. It can accommodate networks with different activation functions and topologies, and it performs high-level reasoning on the network that can curtail the search space and improve performance.
VENUS (Lomuscio, 2023)	It is a neural network verification tool that implements a dependency analysis procedure, thereby pruning the search tree that needs to be considered by Mixed-Integer Linear Programming (MILP)-based formulations of the verification problem and augments it with symbolic interval arithmetic and domain splitting techniques.
whiRL 2.0 (Amir, Schapira, & Katz, 2021)	It is a toolkit to verify complex properties of a DRL agent, using invariant inference techniques to quickly prove properties that could otherwise be quite difficult to verify. It also incorporates abstraction methods for providing some visibility into the DRL system's operation.
Verifiability via Iterative Policy Extraction (VIPER)	It verifies the safety of policies learned through DRL. It focuses on training decision tree policies, which can represent complex behaviors yet are structured enough to be efficiently verified. VIPER employs a novel approach that combines model

Tools	Description
(Bastani, Pu, & Solar-Lezama, 2018)	compression and imitation learning, using a DNN policy as a guide for learning decision tree policies.
Adaptive Stress Testing (AST) (Lee, et al., 2020)	It is a framework designed for finding the most likely path to a failure event in simulation, which is particularly relevant for safety-critical systems like aircraft collision avoidance systems and autonomous cars. The process involves formulating the search for failure events as a sequential decision-making problem. AST can handle both fully observable and partially observable systems.
Matlab Neural Network Verification (Matlab Toolbox for Neural Network Verification, n.d.)	This toolbox implements reachability methods for analyzing neural networks and control systems with neural network controllers in the area of autonomous cyber-physical systems.
Neural Network Enumeration Tool (nnenum-The Neural Network Enumeration Tool, n.d.)	It is a high-performance neural network verification tool. Multiple levels of abstraction are used to quickly verify ReLU networks without sacrificing completeness. Analysis combines three types of zonotopes with star set (triangle) over approximations and uses efficient parallelized ReLU case splitting.

2.2.3.4 Challenges and gaps

In this section a summary of various challenges and gaps for verifying safety of DRL agents is presented. For additional reading refer to:

- *Safe Learning in Robotics: From Learning-Based Control to Safe Reinforcement Learning* (Brunke, et al., 2022)
- *A Study on Overfitting in Deep Reinforcement Learning* (Zhang, Vinyals, Munos, & Bengio, 2018)
- *Deep Reinforcement Learning Verification: A Survey* (Landers & Doryab, 2023)
- *Challenges of real-world reinforcement learning: definitions, benchmarks and analysis* (Dulac-Arnold, Levine, & Mankowitz, 2021)

An investigation on challenges of verifying RL algorithms (van Wesel & Goodloe, 2017) leads both to verification offline and also during runtime, including state-to-action mapping, action sequences, and algorithm properties.

The challenges and gaps for safety verification of RL include the following:

- Many real-world applications where RL can be applied consist of continuous, possibly high-dimensional state and control space, such as using perception sensors for input. This high dimensional space results in computational complexity, making them intractable (Recht, 2019), as well as difficulty in exploration, which is an important aspect of RL algorithms to find optimal policies.
- RL algorithms rely on stochasticity in their environment, which results in high variance in performance from run to run. As a result, stability and convergence of the learning algorithms is a challenge.
- Real-world systems are either expensive to operate or slow for RL algorithms to learn. Therefore, robust policies must be learned from limited samples, requiring the algorithms to sample efficiently.
- Learned policy can be challenging to interpret when leveraging neural networks for function approximation, especially in DRL.
- Safety guarantees during the learning process to prevent the agent from hazardous scenarios may be lacking. These guarantees are critical when the agent is learning in real time and in real-world environments.
- Large neural networks in DRL are highly prone to overfitting because powerful training algorithms often cause the network to memorize specific training data rather than learning generalized policies, mirroring a well-known vulnerability in supervised learning.
- Current techniques for V&V of system and modeling assumptions and performance of DRL agents are limited to small networks, and scalability is challenge.
- Verification of multi-agent systems is an underexplored topic of research.
- It must be validated that the system does the correct thing in the target environment (Nikko, Sjanic, & Heintz, 2021). Under the assumption that the reward function is correctly defined (i.e. captures what is desired), this can be understood as validating the performance.
- New methods are needed for improving the computational efficiency of DRL verification and for guaranteeing tighter bounds when using abstraction-based verification.

2.2.4 Robustness analysis of a collision avoidance neural network

This section of the report presents a case study that assesses a technique for AI/ML V&V. The case study focuses on a neural network designed for collision avoidance in aircraft, which has been trained using PPO RL.

A critical aspect of the verification process for the collision avoidance neural network is understanding its robustness against input perturbations. In this case study, we evaluate the robustness property of the network using a specialized toolbox from Robust Software Engineering group at NASA Ames called SysAI.

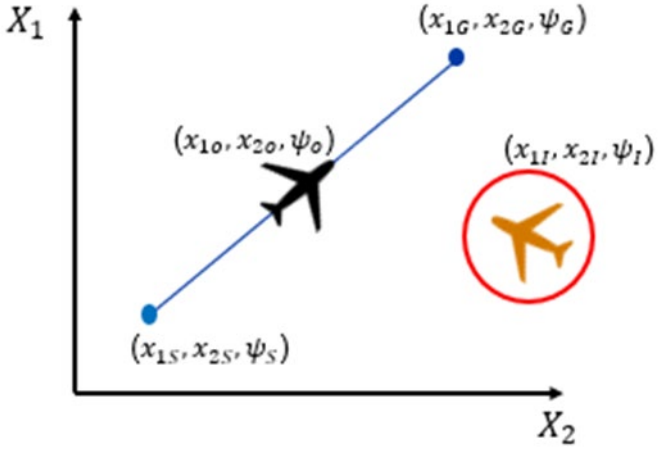

Figure 9. Environment setup for aircraft encounter

The collision avoidance environment setup is shown in Figure 9. The black aircraft represents the ownship aircraft, the orange aircraft represents the intruder. Encounters between the ownship and intruder are obtained from MIT LL Airspace encounter models (MIT LL Airspace Encounter Models, n.d.). Encounter models (EMs) of aircraft is a Bayesian Network trained using aircraft operational data derived from radar or other sensing system flight track data. It provides trajectories for both intruder and ownship aircraft, reflecting realistic aircraft flight dynamics and enabling fast-time simulation.

The EM consists of different models: Manned Correlated, Manned Due Regard, Manned Littoral, Manned Uncorrelated-Conventional, Manned Uncorrelated-Unconventional, Unmanned

Uncorrelated, Unmanned Recreational, UAM Uncorrelated, Terminal Encounter Model. For this case study, Unmanned Uncorrelated EM was selected.

As seen in Section 2.2.1, RL problems are defined using the tuple $\langle S, A, P(s'|s,a), R(s,a), \gamma \rangle$. However, in this case study, the state transition model $P(s'|s,a)$ is not available explicitly, hence a model-free approach is used. Further, as the state space is continuous, function approximation of policy is required to make the problem tractable. The function approximator chosen for this case study is a DNN consisting of two layers of fully connected net with 256 units per layer with tanh activation and a linear output layer.

Ownship aircraft in the simulations are modelled as point mass objects with 2D-Dubins car dynamics. The simulation states available are given by the following vector:

$$o_t = \left[x_{1o_t},\ x_{2o_t},\ \psi_{o_t}, v_{o_t}, x_{1I_t},\ x_{2I_t},\ \psi_{I_t}, v_{I_t}\right]$$

Where $x_{\{1,2\}_t}$ = coordinates of the aircraft at time t, ψ = aircraft heading, v = velocity, the subscript o = ownship, I =intruder. Using this observed state o_t, inputs to the neural network are defined as

$$s_t = \left[\Delta x_{1o_t}, \Delta x_{2o_t},\ \Delta\psi_{o_t}, \Delta x_{1o_{t+1}}, \Delta x_{2o_{t+1}},\ \Delta\psi_{o_{t+1}}, d_{int}, \Delta v_{OI_t}, \Delta\psi_{OI_t},\ d_{goal}\right]$$

Where $\Delta x_{\{1,2\}o_t}$ = deviation of ownship aircraft from trajectory at time step t, $\Delta x_{\{1,2\}o_{t+1}}$ = deviation of ownship aircraft from trajectory at time step $t+1$, $\Delta\psi_{o_t}$ = deviation from desired heading angle, d_{int} =Euclidean distance between ownship and intruder, Δv_{OI_t} = relative velocity between ownship and intruder, $\Delta\psi_{OI_t}$ = relative heading angle between own ship and intruder, d_{goal} = Euclidean distance between ownship and goal. This input state to DNN was defined to ensure that the ownship aircraft avoids collision with the intruder with minimal deviation from the desired trajectory. The output of the DNN is turn rate action $\dot{\psi}$ of the ownship aircraft.

A multi-objective reward function defined for RL training of DNN, given as follows:

$$\mathrm{r}' = -1 - k_u\dot{\psi} - k_d d_{traj} - k_x\left|\Delta_{x_{1t}} + \Delta_{x_{2t}}\right| - k_\psi|\Delta_{\psi_t}|$$

$$r = \begin{cases} r' & ,\ For\ all\ steps \\ r' - k_{int} * R_{collision} & , d_{intruder} \le R_{collision} \\ r' + 1 & , d_{intruder} > R_{collision} \\ k_{goal} & ,\ d_{goal} < R_{goal} \\ r' - k_{end} & , Steps_{sim} \ge Steps_{max} \end{cases}$$

Where $k_{\{u,x,d,\psi,int,end\}}$ = Constants defined in the configuration file and used for tuning the neural network behavior, $R_{\{collision,\ goal\}}$ = distance threshold, $Steps_{\{sim,max\}}$ = Steps taken by the ownship aircraft in the simulation. A penalty of -1 is always applied to r' to minimize the number of steps taken to reach the destination. The control action $\dot{\psi}$ is penalized to prevent jitter in the output. Deviation from trajectory is penalized by both Euclidean and $L_1 - norm$. The $L_1 - norm$ was chosen to provide direction information for the agent. Additional penalties/rewards are added to r' based on conditions. High reward was given to the agent if it reached the goal and high penalty for colliding with the intruder. A penalty was given if the agent did not complete the simulation within the max number of steps. These reward values were tuned/scaled to get desired behavior out of the agent and avoid unintended behavior. Some unintended behavior could result based on the reward selection. If k_{goal} values were low, the aircraft could orbit a fixed point to avoid the intruder. However, setting the k_{end} too high incentivized the aircraft to collide with the intruder to maximize the immediate reward. Thus, initial experimentation before starting the training process is necessary to identify unintended behavior of an agent due to reward hacking.

2.2.4.1 RL training process

The training process for the collision avoidance neural network is shown Figure 10. At first, encounter trajectories are generated in text files using MIT LL Airspace Encounter Models (n.d.) split into test/train set. Ideally in RL the simulation and training environment are tightly integrated, but since EM model was developed in Matlab and RL training was developed in Python, 10,000 text files from the EM model were created and randomly selected during the training.

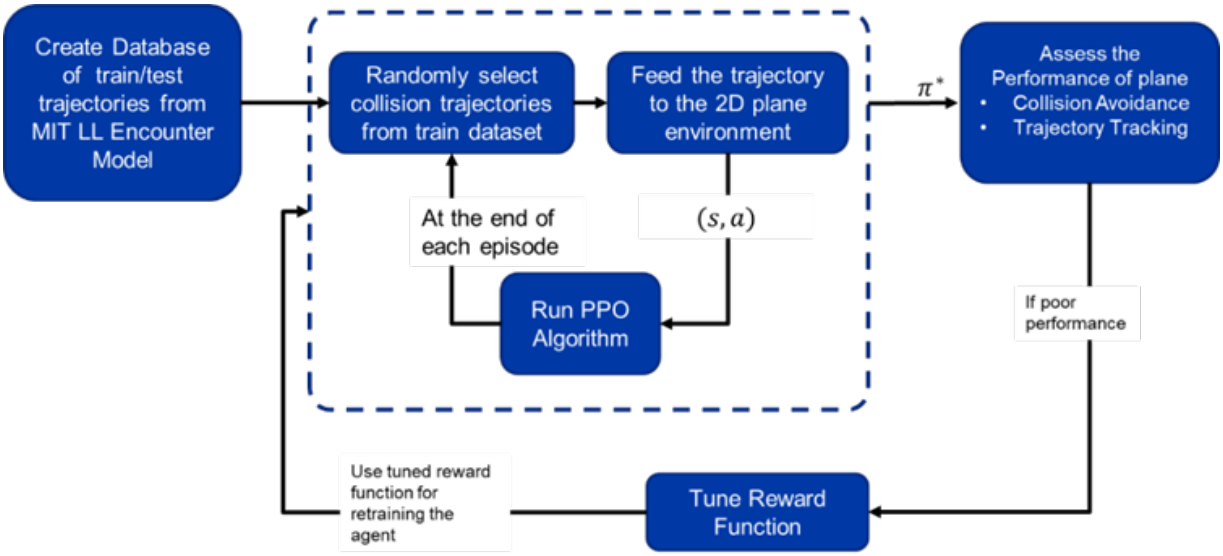

Figure 10. Collision avoidance NN training process diagram

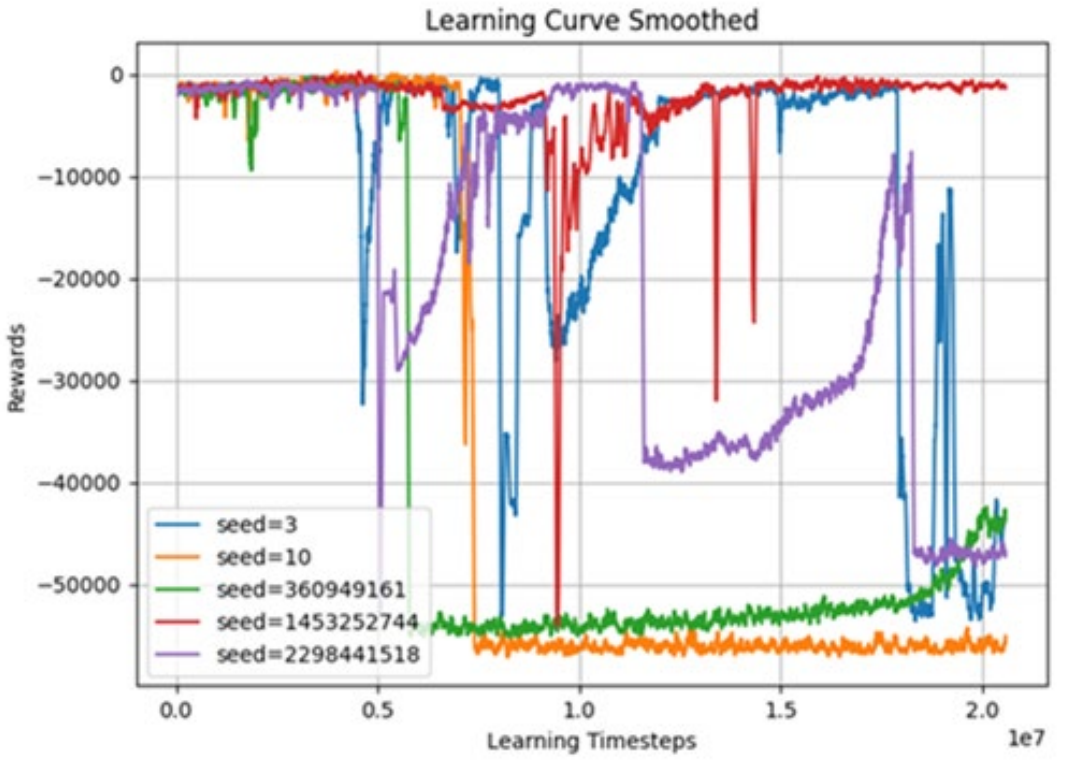

Figure 11. Learning curve for different seed initialization

For training we used the Stable-Baselines3 library (Raffin, et al., 2021) to implement a PPO algorithm. Two layers of fully connected net with 256 units per layer were used for both action and value network. The episodes were terminated if either of the following conditions were satisfied: reached goal, collision with intruder, or reached max steps threshold. Training was conducted for a total timesteps of 20 million. The learning curve for different seed initialization from the training process in shown in Figure 11**.**. The agent overfits and the performance degrades significantly beyond 500,000 timesteps for some of the random seed initialization, and that could be attributed to overfitting to the training samples. But for our final tests, we selected a model with the combined highest cumulative reward and performance metrics, defined in Table 6. It can be seen from the table that seed initialization is an important factor in RL, as it varies the performance of the agent significantly.

Table 6. RL agent training performance metrics

Seed	**Reached Goal(%)**	**No Crash(%)**	**Crash(%)**	**Mean Success Cumulative Reward**	**Mean Max Steps Cumulative Reward**
3	68.5	85.1	14.89	742 ± 45.88	-640 ± 112
10	62.1	90.1	9.9	723 ± 48.18	-614 ± 100.5
360949161	55.6	91.3	8.7	621 ± 52.38	-818 ± 333.76
1453252744	60.19	85.1	14.89	751 ± 27.97	-548 ± 75.96
2298441518	41.6	89.7	10.29	729 ± 34.78	-591 ± 86.85

A sample of collision avoidance maneuvers executed by the trained agent is shown in Figure 12. The orange trajectory represents the initial trajectory that leads to collision with intruder (circled in red). The trained agents generate the blue trajectory to avoid collision. In both scenarios, the agent was able to reach the goal within the defined threshold distance.

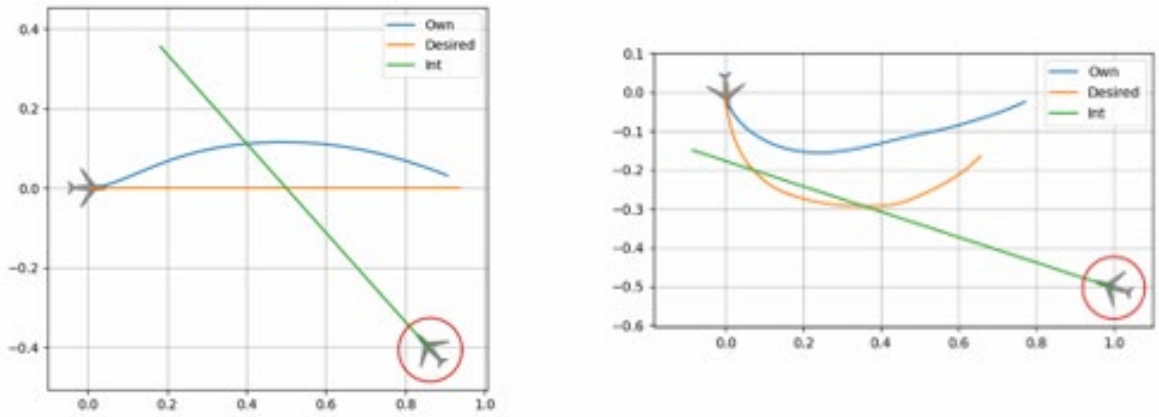

Figure 12. Trained collision avoidance neural network

2.2.4.2 Overview of SysAI

SysAI is a flexible statistical learning framework and toolbox for the V&V and analysis of complex and high-dimensional aerospace systems with DNN components, developed in NASA Ames Robust Software Engineering (RSE) Group. SysAI connects to the complex system being studied and its simulators using a small customizable interface, enabling the user to analyze a wide variety of systems written in C/C++, Python, Java, R, Simulink/Matlab, or a combination thereof.

SysAI provides functionality for many important V&V tasks, including:

- Statistical analysis of training and test data: a detailed statistical analysis of the data sets used for training and evaluation of the neural network
- Safety-envelope analysis: automatic analysis of the safety-envelope in high-dimensional state space, which indicates under which operational conditions the system is behaving safely
    - Geometric shapes modeling identifies and characterizes regions with similar behavior and describes these regions in geometrical terms. SysAI thus helps to improve explainability of the system.
- Property checking: supports automatic checking and analysis of safety and performance requirements

- Time-series analysis: performs advanced time-series analysis in a high-dimensional parameter and state space

Figure 13 shows the high-level system architecture and the results of the simulations reported by He, Schumann, & Yu (2022) and He & Schumann (2023). The System under Test (SuT) and its simulators, which can be implemented in Python, Matlab, C/C++, R, or a combination of these is connected to SysAI using a small API. During the experiment, SysAI produces a vector of parameter values under which the SuT is executed. For the safety envelop analysis, for example, the behavior of the SuT is analyzed by SysAI to intelligently produce an efficient exploration of the high-dimensional parameter space and to characterize boundaries separating safe from unsafe regions.

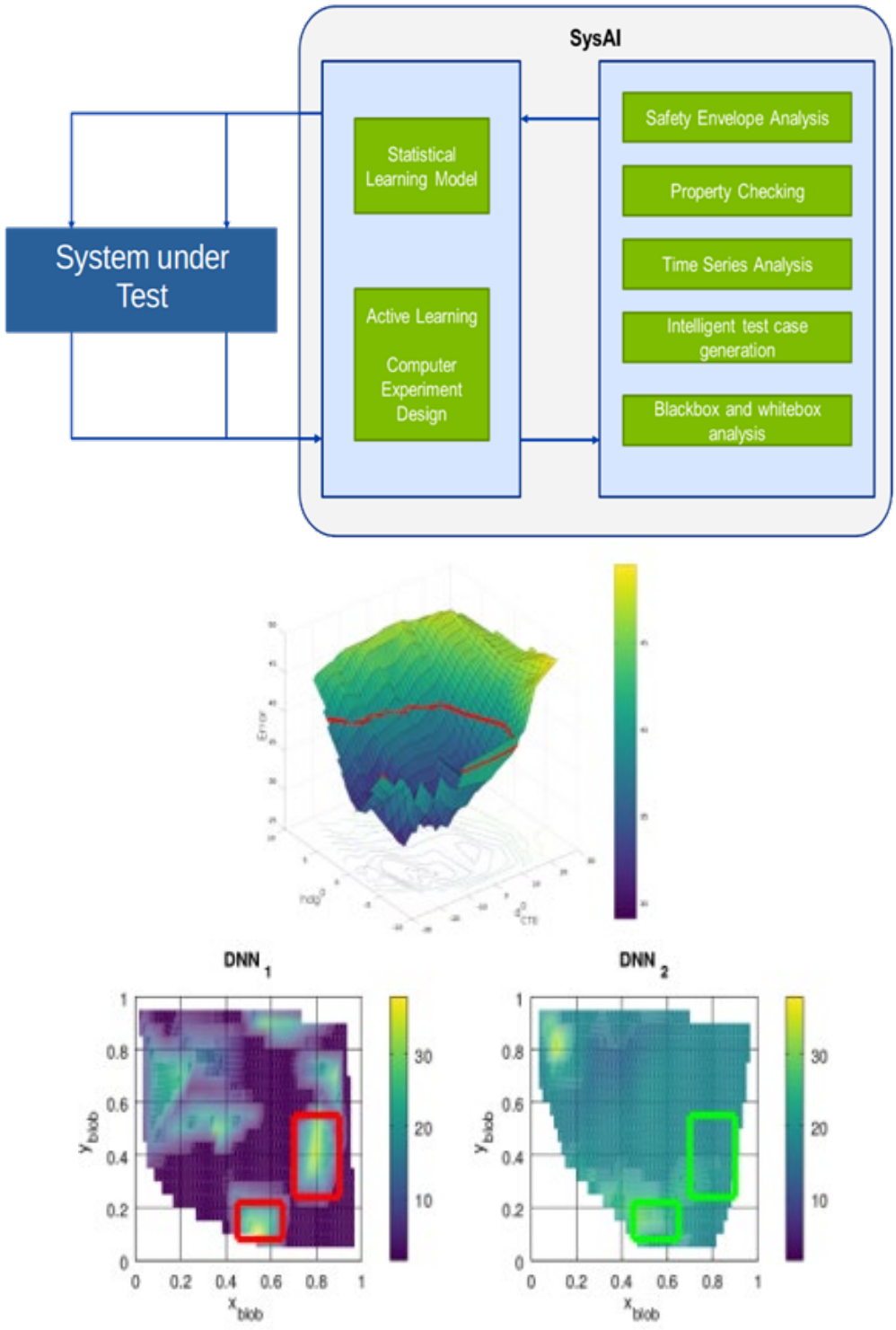

Figure 13. High-level system architecture of SysAI and SuT simulation reports

2.2.4.2.1 Collision avoidance neural network integration into SysAI

The collision avoidance neural network is integrated with SYSAI in a straightforward manner: for analysis, the SuT consists of the collision avoidance neural network and the aircraft

environment, which is in charge of performing aircraft kinematics for the ownship, calculating distances to the intruder, and determining if the current scenario will end successfully or not. The SuT is to a large extent based on the code for analyzing the training effort as described above.

For analysis using SysAI, the SuT will be driven by a list of parameters, which are to be explored. Typical examples include speed of the ownship, offsets to initial position and heading, or quality of position estimates, as well as environmental parameters e.g. weather, wind, etc.

An experiment with SysAI is controlled by an experiment configuration file that contains all necessary information, like number of runs, ranges of parameters, and definition of the interface. While running, SYSAI explores the parameter space by generating parameter sets that will be evaluated by a run of the SuT. The SuT will return if that specific point in the state space is safe or not. SYSAI's active learning algorithms will use this information to explore the high-dimensional parameter spaces efficiently, in particular for the safety envelop analysis, the boundaries between safe and unsafe regions in high-dimensional space. All resulting data are saved in flexible file format for subsequent analysis and results visualization, generation of figures, etc.

2.2.4.2.2 Results from robustness analysis

In this work, we used SysAI to carry out a set of experiments for the robustness analysis of the collision avoidance neural network. In particular, we were interested in how the perturbations to the state of the ownship and control command affect the performance of neural network. Further, the robustness of the neural network was also assessed in different encounter categories as defined in DAA MOPS in RTCA DO-365B (RTCA, Inc., 2021), particularly converging from left or right.

Given a specific parameter setting for ownship, the performance of the collision avoidance neural network was evaluated, as described above, by running multiple trajectories from the data set that has been produced via the MIT trajectory generator. SysAI experiments are set up to explore high-dimensional parameter spaces. For better visualization, the following experiments are projected onto two parameters space.

In the first experiment, we offset the initial position of the ownship in x and y direction. Since all experiments are carried out with normalized coordinates in [0,1], our parameterized offset is applied as: x_1=x_traj^1+off_(x_1) and x_2=x_traj^2+off_(x_2) where (x_traj^1 ,x_traj^2) is the starting point of the ownship trajectory. The offsets were selected in the range of [-0.2,0.2]. The resulting starting point is then propagated throughout the run using the two-dimensional aircraft kinematics.

Each parameter setting was evaluated over M=10 randomly selected MIT trajectories. The parameters were considered successful if more than 65% of the runs did not lead to a crash. They were considered failed if more than 35% lead into crashes or non-termination. This threshold is used as a placeholder which can be updated easily according to actual requirement.

The active learning algorithm of SysAI is initialized by 10 randomly selected parameter settings (shown as 'x' in the plot), followed by N=50 runs guided by active learning (circles). Successful runs are shown in blue, failure runs are shown in red. The axes show the offset with respect to x_1 (horizontal) and x_2 (vertical).

When considering all categories of trajectories of Figure 14, all collision scenarios (left), converge from left (middle), converge from right (right), the analysis shows that the collision avoidance neural network is robust in an area around the nominal values i.e., when both offsets are (0,0). The area of acceptable performance, which is shown as the blue safety region is of an elongated shape and clearly separated by the red failure region. The results indicate that the neural network is more sensitive with respect to disturbances of the x1 values.

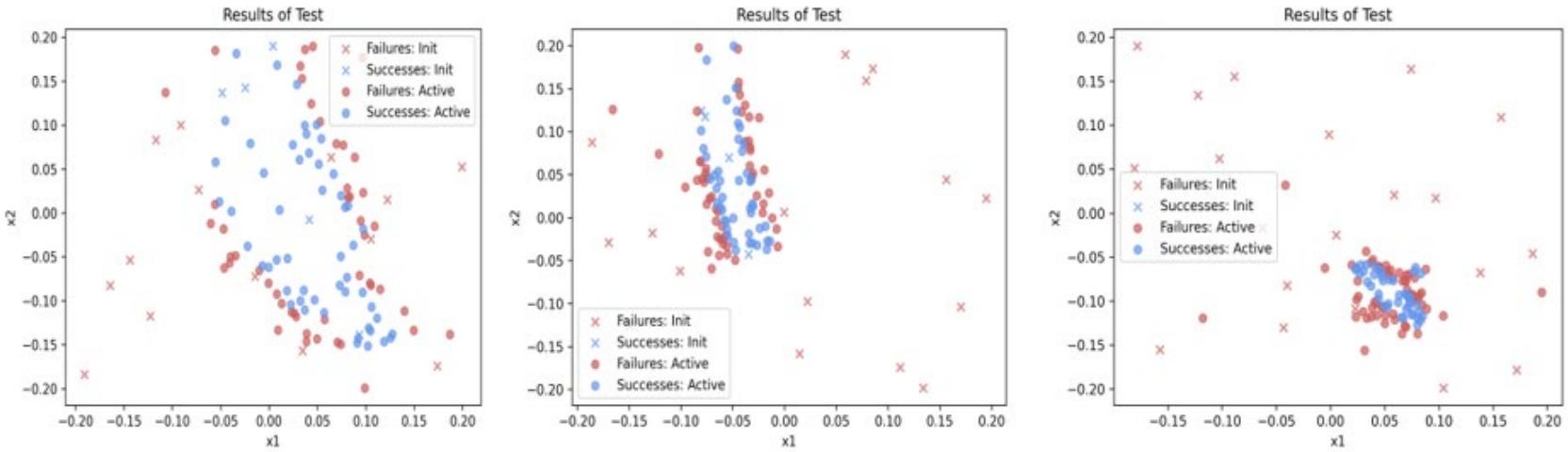

Figure 14. All collision scenarios(left), converge from left (middle), converge from right (right)

All collision scenarios (left), converge from left (middle), converge from right (right) indicate clearly marked safety regions. Due to the geometric situation, such regions only show up for negative offsets with respect to x1 for "left convergence." and positive offsets for "convergence from right."

In these experiments, only two categories have been used, because the number of MIT trajectories provided for the other collision categories as defined in DO-365 was too small. We also only perturbed the initial position of the ownship in these experiments. The positions of the intruder AC cannot be changed easily, because the test driver only provides AC dynamics for the ownship.

We also carried out experiments with other parameter variations.

The steering command u_{cmd}, which controls the ailerons, and which is derived from the neural network output, might be noisy. We modeled this behavior by adding Gaussian noise N(μ,σ). As expected, the system behavior deteriorates with larger μ and σ. Figure 15 (right) shows this roughly elliptic "success" area; the horizontal axis shows μ, the vertical axis log(σ). μ was modified in the range of [-0.2, 0.2], and σ in the range [0.001, 1].

The speed of the ownship v_{own} can have a big impact of the outcomes of the collision avoidance. Figure 15 (left) shows an experiment where the command output u_{cmd} was modified by +-20% (vertical axis) and the speed changed to 83% to 117% of nominal (horizontal axis). Again, the safety region around nominal conditions is clearly separated from the unsafe region, characterized by red data points.

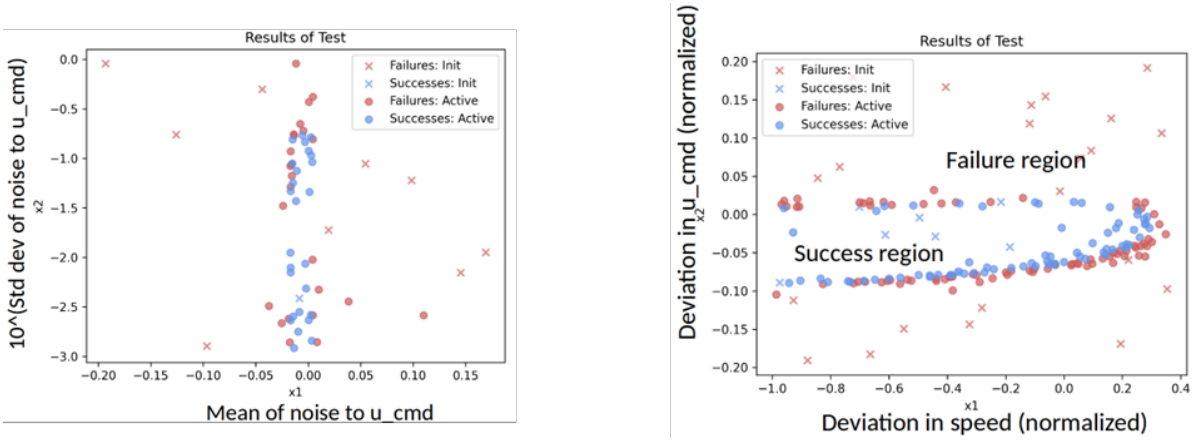

Figure 15. (Left) Success and failure regions for noisy command; (Right) Success and failure regions defined by variation in speed and control command

2.2.4.3 Conclusion

This chapter of report presents the development of a collision avoidance neural network for aircraft for horizontal encounters. A similar approach could be undertaken to extend this to three-dimensional collision avoidance scenarios. Various challenges during reward modeling and training were presented with potential solutions. For V&V, robustness analysis was conducted for neural networks with perturbations to state and control commands using the SysAI toolbox.

In general, multidimensional analysis does not scale well (the "curse of dimensionality"), because the number of experiments might grow exponentially. But SysAI's advanced statistical learning/modeling methods as well as its Bayesian optimization techniques are designed for and can help address the scalability issue in high-dimensional analysis efficiently.

For example, SysAI can reduce the number of test runs of the SuT that need to be carried out for robustness analysis. In the safety envelope analysis, it uses Bayesian dynamic tree and particle filtering modeling, as well as a search metric technique that has been specifically designed to search in the vicinity of safety boundaries rather than exploring the entire space. Therefore, a better performance than current implementations of ALS/ALM or expected improvement has been achieved in many cases. Whereas the other optimization goal is to explore areas in a fairly generic manner, our metric is focusing on boundary detection. This is just one example. Other techniques include advanced sensitivity analysis to identify the most affecting parameters in the high-dimensional space, both for explainable dimension reduction as well as significant scalability improvement. The actual performance strongly depends on the SuT. SysAI does not limit the number of dimensions, and the identification of promising dimensions in a preprocessing step can speed up analyses considerably.

2.3 Reproducibility issues of PyTorch-trained models

This section examines the reproducibility of ML models that are trained with PyTorch, currently the most commonly used ML coding platform. A model is reproducible if every time the training algorithm is repeated using the same dataset and the same setting, the identical resulting model is produced.

However, from the PyTorch official documentation (PyTorch, n.d.), it is stated that "completely reproducible results are not guaranteed across PyTorch releases, individual commits, or different platforms. Furthermore, results may not be reproducible between central processing unit (CPU) and graphics processing unit (GPU) executions, even when using identical seeds."

According to this statement, PyTorch trained models are not completely reproducible if you retrain a model with a different PyTorch release and even on a on a different computer. PyTorch provides some instructions to limit the randomness during training that involve setting seeds for all random generators and avoiding using nondeterministic algorithms. These settings can significantly affect the performance of a trained model, but they still do not completely eliminate the randomness.

PyTorch, however, does not mention whether it guarantees reproducibility of a model if retrained on the same platform using the same PyTorch release. There are online sources, such as *Issues in the reproducibility of deep learning results* (Jean-Paul, Elseify, Obeid, & Picone, 2019), that say that between-runs reproducibility is not guaranteed even for the same PyTorch release and platform. In this work we conducted experiments to verify this issue and get more insight into the variability of Pytorch-trained models and any impact it may have on their performance.

2.3.1 Experiments

As an example, we used the PyTorch implementation of the YOLOv4 algorithm for object detection (Bochkovskii, n.d.)]. In the experiment we trained the YOLOv4 model on the COCO object detection dataset (cocodataset, n.d.). Specifically, we trained six model instances on this dataset using the GPU and the same parameter setting. The six models were trained using the same computer with the same NVIDIA GPU GeForce GTX TITAN X and PyTorch 1.7.1, CUDA 10.1. As the computer has multiple GPUs, we always trained on GPU 0.

We followed all PyTorch instructions to set a seed for every random generator used and avoided using nondeterministic algorithms. Specifically, the following piece of code was executed in the beginning of the training algorithm:

```
def set_seed(seed: int = 42):
      np.random.seed(seed)
      random.seed(seed)
      torch.manual_seed(seed)
      torch.cuda.manual_seed(seed)
      # When running on the CuDNN backend, two further options must be set
      torch.backends.cudnn.deterministic = True
      torch.backends.cudnn.benchmark = False
      # Set a fixed value for the hash seed
      os.environ["PYTHONHASHSEED"] = str(seed)
```

```
print(f"Random seed set as {seed}")
```

In addition, we used a single thread for data loading. With these settings, each model instance was trained for 300 epochs and then tested on the same COCO validation set. The six resulting models are then compared in terms of their parameters as well as performance.

2.3.2 Parameter variability

YOLOv4-Tiny has 118 learnable modules of two types: Conv2d and BatchNorm2d, that have in total 6,062,833 learnable weight parameters. We found that the values of these parameters are different among the six trained model instances, which confirms that PyTorch GPU-trained models are not completely reproducible even for the same PyTorch release and hardware platform. To measure the variability, for each parameter we computed the mean and standard deviation (std) across the six trained models, and the relative std as follows:

$$Std_{relative} = std/|mean|$$

The relative std was computed for each module and then averaged over all the modules. For stability, we ignored the parameters whose mean was smaller than 1% of the mean value over the whole module. Table 7 below shows the average and maximum relative std obtained as the training progressed.

Table 7. Parameter variations statistics

Epoch	0	24	274	299
Average Relative Std (%)	56	174	247	257
Max Relative Std (%)	307	656	1062	1220
Module with max std	4.BatchNorm2d	18.Conv2d	4.BatchNorm2d	4.BatchNorm2d

As observed, the variation of parameter values among the model instances is significant, as the average relative std is 56% after just the first epoch.

2.3.3 Performance variability

The performance of each model is evaluated on the COCO validation set using the standard mean average precision (mAP) and presented in Table 8.

Table 8. Performance variability in different models

Model	M1	M2	M3	M4	M5	M6
mean average precision (mAP) (%)	36.5	36.0	36.6	36.0	36.2	36.7

We can see that despite the significant variation in parameter values, the difference in the performance of the models is insignificant.

2.3.4 Conclusion

In conclusion, the following were the findings of this empirical analysis on reproducibility using PyTorch:

- The experiments showed that early releases of PyTorch (<=1.10) and CUDA (<=11.3) do not guarantee between-runs reproducibility, even on the same PyTorch release and hardware platform.
- Despite the significant variation in parameter values, the difference in the performance of models trained with the same setting is insignificant.
- Recent releases of PyTorch and CUDA may provide between-runs reproducibility. This needs to be confirmed with further experiments.

Recent experiments with a Resnet50 model, which is much simpler than YoloV4, were able to reproduce the model on PyTorch 1.14 and CUDA 12.0. The same Resnet50 model, however, was not reproducible between runs if trained with PyTorch 1.10 and CUDA 11.3. This suggests that new versions of PyTorch or CUDA may improve in terms of reproducibility. As future work, further experimentation of YOLO with newer PyTorch releases and CUDA can be performed to evaluate the trained model reproducibility.

2.4 Evaluation of TensorFlow ML deployment pipeline

ML algorithms offer great capabilities for object detection and image segmentation and classification. They are increasingly improving in accuracy and latency for these tasks. However, less attention has been paid to their hardware deployment.

Some of the important considerations made during the deployment of ML models are:

- Latency (there is no round trip to a server)
- Privacy (no personal data leaves the device)
- Connectivity (internet connectivity is not required)
- Size (reduced model and binary size)
- Power consumption (efficient inference and a lack of network connections)

Optimizing the ML models for any of these factors has a significant impact on their inference performance. Thus, it is critical to examine the deployment pipeline and evaluate the variation in performance of ML models optimized for hardware deployment.

In this section, a popular machine learning framework, TensorFlow (TF) (Abadi, et al., 2015), is explored for hardware deployment. The impact of various optimization techniques in TF are evaluated with a case study using a version of YOLOv4 in TF. Benchmarking results from this case study clearly indicates that the performance of YOLOv4 is affected significantly by the different deployment techniques available in TF.

2.4.1 TensorFlow deployment pipeline

A typical machine learning pipeline includes data processing, model training, evaluation, deployment, and continuous improvement. Since the focus of this section is ML model optimization and its deployment, the pipeline explored using TF is the one shown in Figure 16. The deployment pipeline assumes a trained/frozen model is available as a saved model in TF protobuf format (*.pb).

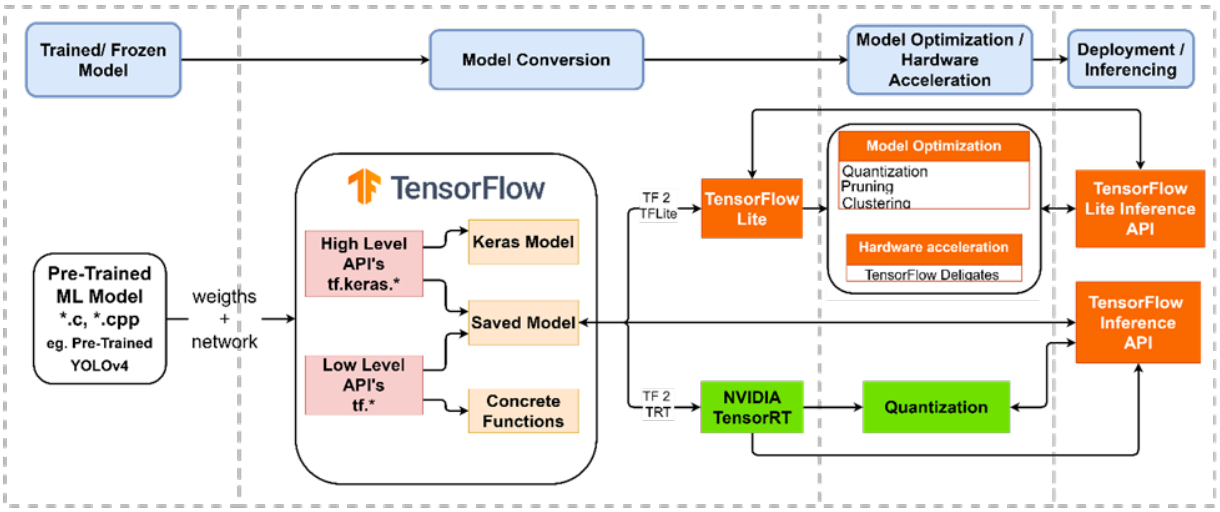

Figure 16. TensorFlow ML deployment pipeline

The saved model is used to explore three deployment branches. The first branch involves deploying the saved model using TF's inference engine. This method is favorable when no model optimization is required for deployment. There exists a two-way arrow between the saved model and TF inference API, which indicates that if any model optimizations are required, they

need to be applied to the model itself. As of the time of writing this report, TF did not have any capabilities for direct saved model optimizations.

The second branch explores deploying the saved model by converting it to TensorFlow Lite (TFLite) (TenorFlow Lite Conversion, 2022). This option is selected when deploying the ML model on an embedded system, mobile, or edge device. Various optimization options are available when converting model from TF to TFLite. These optimization options result in reduced latency and reduced memory footprint. They also enable execution on hardware restricted or optimized for fixed-point operations or optimize the model for special purpose hardware accelerators.

The third branch explores TF integration with TensorRT (NVIDIA, 2018-2026). TensorFlow-TensorRT (TF-TRT) is a deep-learning compiler for TF that optimizes TF models for inference on NVIDIA devices. TF-TRT is the TF integration for NVIDIA's TensorRT (TRT) High-Performance Deep-Learning Inference Software Development Kit (SDK), allowing users to take advantage of its functionality directly within the TF framework. TF-TRT also offers quantization of parameters to improve the performance of inference.

2.4.1.1 TensorFlow Lite model optimization

TFLite model optimizations options enable running inference on hardware with constraints while preserving its original performance (TensorFlow Model Optimization, n.d.). There are several benefits of applying optimization, such as size reduction, latency reduction, accelerator compatibility. However, there exist some trade-offs as well. Optimizations can potentially result in change in model accuracy, so attention must be paid while applying these techniques.

TFLite offers the following optimization techniques:

- *Quantization:* Quantized models have lower precision, such as 8-bit integers as opposed to default 32-bit float. Lower precision is a requirement to leverage certain hardware. Quantization can be applied after or during training.
- *Sparsity and pruning:* Sparse models are those where connections between operators (i.e. neural network layers) have been pruned, introducing zeros to the parameter tensors.
- *Clustering:* Clustered models are those where the original model's parameters are replaced with a smaller number of unique values.
- *Collaborative optimization:* The TF toolkit provides experimental support for collaborative optimization. This enables combining several model compression

techniques and simultaneously achieving improved accuracy through quantization aware training.

There also exist options for improving latency through hardware delegation, but they currently do not support NVIDIA GPUs. Therefore, hardware delegation is not explored in this report, and only post-training quantization is explored.

Post-training quantization includes general techniques to reduce CPU and hardware accelerator latency, processing, power, and model size with little degradation in model accuracy. These techniques can be performed on an already-trained float TF model and applied during TFLite conversion. The options available for post-training quantization during conversion from TF to TFLite are summarized in Table 9 (TensorFlow Model Optimization, n.d.). With post-training quantization, it is expected that the model size reduces by ~50-75% with little loss in accuracy. No additional data is required for quantization except when performing integer quantization.

Table 9. Summary of various quantization techniques in TensorFlow Lite

Technique	**Data requirements**	**Size reduction**	**Accuracy**	**Supported hardware**
Post-training float16 quantization	No data	Up to 50%	Insignificant accuracy loss	CPU, GPU
Post-training dynamic range quantization	No data	Up to 75%	Smallest accuracy loss	CPU, GPU (Android)
Post-training integer quantization	Unlabeled representative sample	Up to 75%	Small accuracy loss	CPU, GPU (Android), EdgeTPU, Hexagon DSP

The quantization scheme works by reducing the precision of activations and weights of the ML model. Table 10 summarizes precision conversions options, when appropriate post-training quantization is selected (TensorFlow Model Optimization, n.d.). In dynamic range, quantization converts the weights to 8-bit precision, and for operations that support quantized kernels, the

activations are quantized to 8-bit precision dynamically prior to processing and are de-quantized to float precision after processing.

Table 10. Summary of parameter conversion for various quantization options in TensorFlow Lite

Post-training Quantization		**float32**	**float16**	**Int8**	**int16**
Dynamic Range,	float32	✓ {activations}		✓ {weights, activations}	
Float 16 Quantization,	float32		✓ {weights, activations}		
Strict Integer,	float32			✓ {weights, activations}	
Float Fallback,	float32	✓*{weights, activations}		✓ {weights, activations}	
Integer16,	float32			✓ {weights}	✓ {activations}

Strict integer quantization option converts the 32-bit floating point weights and activation outputs to nearest 8-bit fixed integer. The integer quantization is performed as per the method described by Jacob, et al. (2018). Each layer of the quantized parameter is based on quantization level and clamping range. The quantization operation is performed point-wise using the function q shown below:

$$clamp(r; a, b) = \min(\max(x, a), b)$$

$$s(a, b, n) \coloneqq \frac{b - a}{n - 1}$$

$$q(r; a, b, n) \coloneqq \left\lfloor \frac{clamp(r; a, b) - a}{s(a, b, n)} \right\rceil s(a, b, n) + a$$

Where $r =$ real valued input, $[a, b] =$ quantization range, $n =$ fixed quantization levels for all layers $= 2^8 = 256 = 8 - bit$ quantization and $\lfloor . \rceil$ denotes rounding to the nearest integer. The values of a, b, s are obtained from unlabeled representative dataset. TFLite supports per-axis

quantization, i.e. scale and ranges are obtained per slice of the tensor. The weights are forced to be symmetric and have zero-point equal to 0. However, the activations are asymmetric and can have their zero-point anywhere within the signed int8 range $[-128,127]$. TFLite also supports converting the activations to 16-bit integer and weights to 8-bit integer during model conversion. This mode can improve accuracy if the activation is sensitive to quantization.

The Float Fallback option is similar to integer quantization, where the weights and activation outputs are quantized. The only difference in Float Fallback is that processes not supported by TFLite's quantization optimization are left in FP32 precision. This strategy allows conversion to complete, resulting in a smaller and more efficient model. However, this model will not be compatible with integer-only hardware.

2.4.1.2 TensorFlow with TensorRT optimization

NVIDIA TensorRT is a C++ library that facilitates high performance inference on NVIDIA GPUs in lower precision Float16 and Int8 (NVIDIA, 2018-2026). TensorRT takes a trained network, which consists of a network definition and a set of trained parameters, and produces a highly optimized runtime engine which performs inference for that network.

TensorFlow has an integration with the TensorRT library, enabling optimization of compatible TF graph elements of a ML model and executes it on NVIDIA GPU. Noncompatible TF subgraph computations are handled by TF. The graph optimizations do not change the underlying model of the graph. They look to restructure the graph to perform operations faster and efficiently on NVIDIA GPUs. The graph optimization process occurs in three phases: graph partition, layer conversion and engine optimization, as shown in Figure 17 (Davoodi, Lai, Morris, & Sharma, 2019).

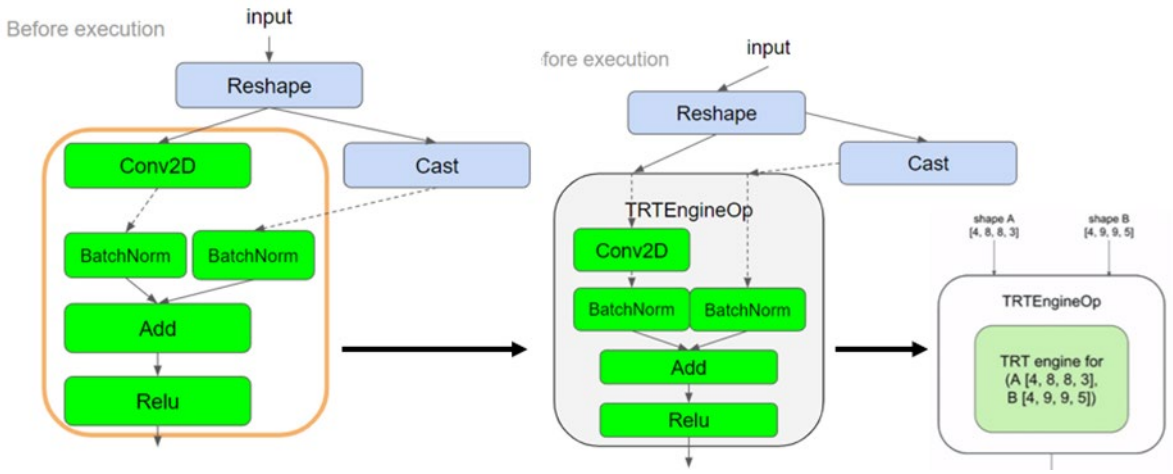

Figure 17. An example subgraph formation stages using TensorFlow-TensorRT Optimization

During graph partition, TRT identifies compatible vs. noncompatible graph elements. After the first phase, it wraps the compatible elements in a subgraph, forming a single node. The subgraph formation is constrained to be a direct cyclic graph and have no loops. In the second phase for each new node, a TensorRT network is built. Finally, in the last phase the TensorRT network is optimized to build a TensorRT engine. Additional details about the TensorRT optimization can be found in *Acceslerating Inference in TensorFlow with TensorRT User Guide* (NVIDIA, 2018-2026) (Davoodi, Lai, Morris, & Sharma, 2019).

2.4.2 Case study

To assess the different optimizations techniques available in TensorFlow, a case study was conducted. YOLOv4, a popular object detection ML model, was chosen for this case study. The following sections describe the case study methodology and summarize the results.

2.4.2.1 Machine learning for object detection

DNNs, a particular category of ML algorithms, have gained a lot of traction in the field of computer vision for object detection and classification. A DNN model for object detection defines a bounding box around the object and categorizes it based on predefined classes. The bounding box is obtained by regression and defines the coordinates of the top left or bottom right vertices, along with the width and height of the box. Categorization of the object inside the bounding box is based on a typical classification methodology in ML and defines the probability of the object falling into a particular class.

A crucial component of DNN for computer vision is multilayer convolutions (Szeliski, Computer vision: algorithms and applications, Second Edition, 2022). When convolutions are used in a

DNN, they are referred to as *convolutional neural networks* (CNNs). In CNNs, instead of connecting all of the units in a layer to all the units in a preceding layer, convolutional networks organize each layer into feature maps, which can be inferred as parallel planes or channels. In a convolutional layer, the weighted sums are only performed within a small local window, and weights are identical for all pixels.

The DNN/CNN networks used for object detection are broadly classified as one-stage or two-stage detectors based on their meta-architectures, as shown in Figure 18 (Carranza-García, Torres-Mateo, Lara-Benítez, & García-Gutiérrez, 2021). Two-stage detectors divide the detection process into the region proposal and the classification stage. These models first propose several object candidates, known as regions of interest (RoI), using reference boxes (anchors). In the second step, the proposals are classified, and their localization is refined. On the other hand, one-stage detectors contain a single feed-forward fully convolutional network that directly provides the bounding boxes and the object classification. In the following section, a one-stage object detector will be described.

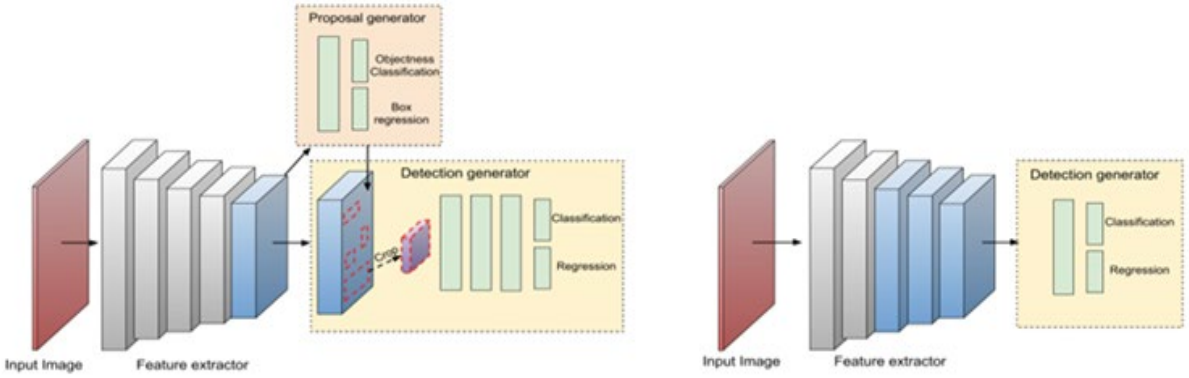

Figure 18. General categorization of deep learning based object detection meta-architectures

2.4.2.2 YOLOv4 model

A modern object detector usually consists of a backbone, neck, and head for prediction, as shown in Figure 19. The backbone extracts features from multiple resolutions. The neck is a layer between the head and backbone that collects feature maps from different stages of the backbone and performs multiresolution feature aggregation. Finally, the head is used to predict the bounding and class prediction.

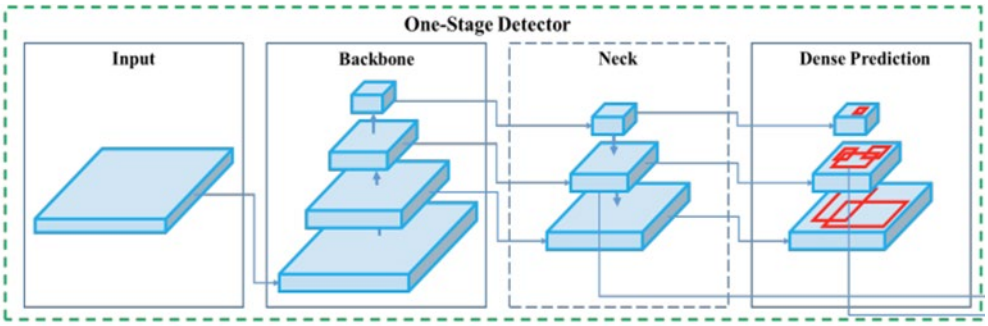

Figure 19. YOLOv4 object detector architecture

YOLOv4 (Bochkovskiy, Wang, & Hong-Yuan, 2020) is a real-time CNN-based one-stage object detector, build with the following components:

- Backbone: CSPDarknet53
- Neck: Spatial Pyramid Pooling (SPP), Path Aggregation Network (PAN)
- Head: YOLOv3

Bag of Freebies (BoF) are training strategies to increase accuracy without increasing inference cost, and Bag of Specials (BoS) are plugin modules and post-processing methods that only increase the inference cost by a small amount but can significantly improve the accuracy of object detection. BoF and BoS are used to improve the performance of YOLOv4 over YOLOv3. They are as follows:

- *BoF for backbone*: CutMix and Mosaic data augmentation, DropBlock regularization, class label smoothing
- *BoS for backbone*: Mish activation, cross-stage partial connections (CSP), Multi input weighted residual connections (MiWRC)
- *BoF for detector*: CIoU-loss, CmBN, DropBlock regularization, Mosaic data augmentation, Self-Adversarial Training, eliminate grid sensitivity, using multiple anchors for a single ground truth, cosine annealing scheduler (Nasrudin, Iriawan, Fithriasari, Pravitasari, & Hidayat, 2022), optimal hyperparameters, random training shapes
- *BoS for detector*: Mish activation, SPP-block, Spatial Attention Module (SAM)-block, PAN path-aggregation block, Distance Intersection over Union Non-Maximum Suppression (DIoU-NMS)

2.4.2.3 Performance assessment

For ease of integration, a version of YOLOv4 in TF (TensorFlow YOLOv4, 2020) was opted for performance assessment case study. Different post-training quantized models of YOLOv4 in TFLite and TF-TRT were used and compared with the baseline TF model. Three metrics were used for gauging the performance of different models:

- mAP with Intersection of Union (IoU) value of 0.5
- Inference Time(s) – time to perform inference on a single image
- Model Size (MB)

The Microsoft COCO dataset (1000 images) was used for evaluation. 100 unlabeled images of the same dataset were used with quantization options when required. To prevent any discrepancies caused due to different versions of TF, Docker containers from TF and NVIDIA were used for the case study. The docker containers were executed on two configurations, as shown in Table 1-6 .

Table 11. System Configuration

Dell Workstation	**Jetson Nano**
GPU: NVIDIA GeForce GTX 1080, 8GB	**GPU:** NVIDIA Tegra, 128-core NVIDIA Maxwell™ architecture-based GPU
CPU: Intel(R) Xeon(R) CPU E5-1660 v4 @ 3.20GHz, x86_64	**CPU:** Quad-core ARM® A57
Memory: 128 GB	**Memory:** 4GB 64-bit LPDDR4; 25.6 gigabytes/second
OS: Ubuntu 20.04.4 LTS	**OS:** Linux for Tegra®, Ubuntu 18.04
TensorFlow v 2.9.1, 2.3	TensorFlow v2.7

The results from performance assessment of different quantized versions of YOLOv4 on the two systems are shown in Figure 20 and Figure 21 below. The baseline model refers to the YOLOv4 model in TF. When the YOLOv4 TF model is quantized post-training in TFLite, additional nodes are added to the original graph of the model, as shown in Figure 20. These additional nodes quantize the data for supported nodes and dequantizes the data for nodes not supported by TFLite for quantization.

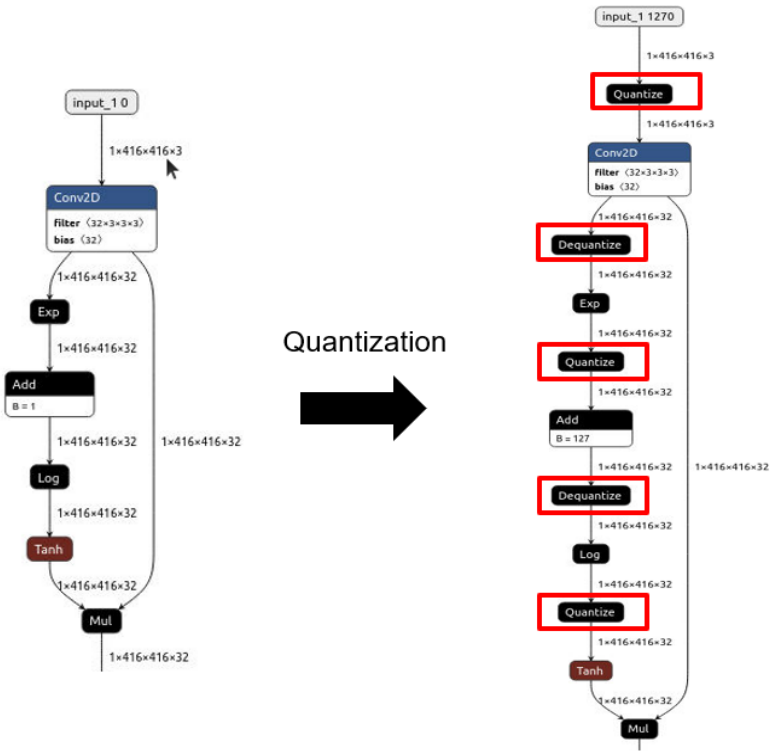

Figure 20. Small section of YOLOv4 after TFLite quantization optimization

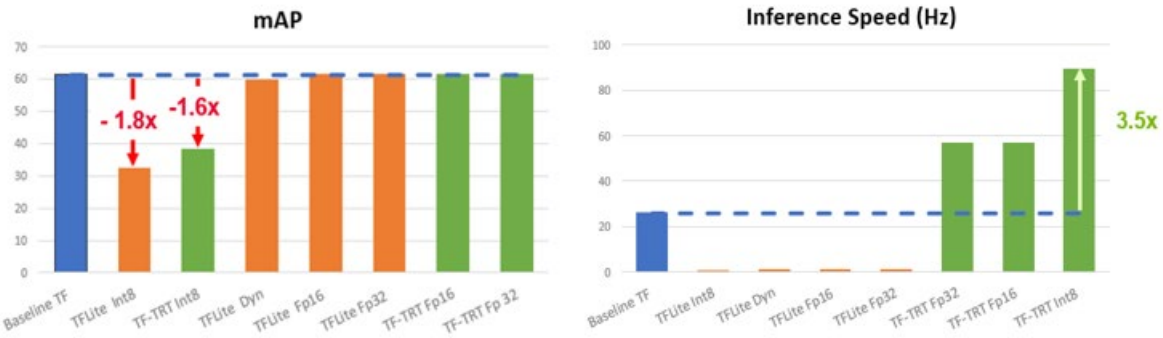

Figure 21. mAP and inference speed for different YOLOv4 quantized models on a Dell Workstation

As expected, the TFLite quantized models have significant reduction in model size compared to original model (shown as blue bar) in Figure 22. The size of TFLite Float 16 precision and Int8 precision are ~50% and ~25% of baseline model size, respectively. However, the TF-TRT models have a ~130% and ~340% increment in model size for Int8 and FP16 precision respectively. This

tremendous increase in model size for TF-TRT could be attributed to saving unused variables. It is expected to resolved in latest version of TensorRT.

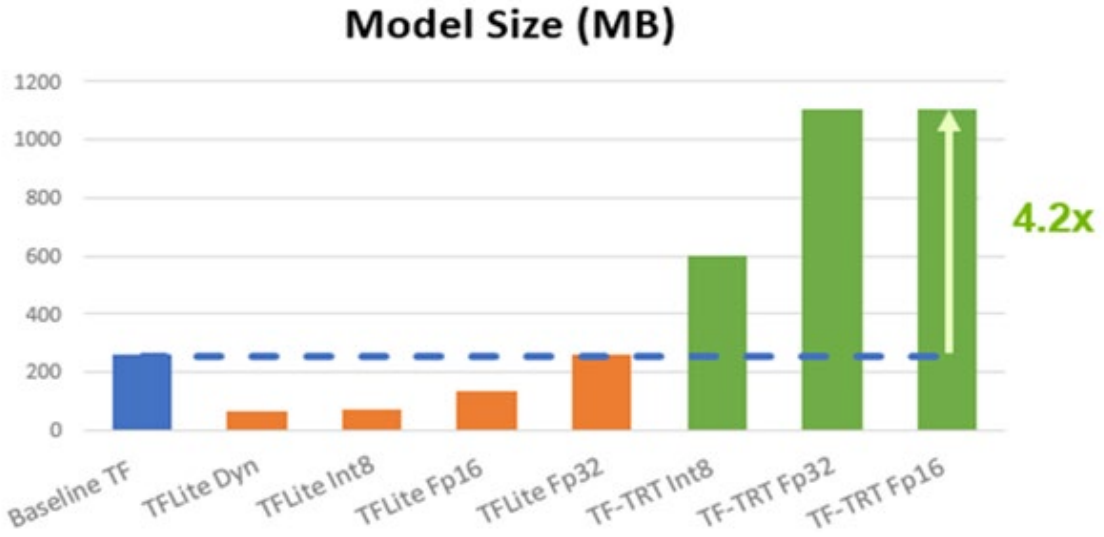

Figure 22. Size of different quantized versions of YOLOv4

Comparison of mAP and inference time for different YOLOv4 quantized models on Dell Workstation is shown in Figure 21. The baseline model, shown in blue, is in TF with FP32 precision for weights and activation outputs and utilizes the NVIDIA GPU available on the workstation. As seen in the Figure 21, TFLite and TF-TRT Int8 precision quantized models have the highest drop in mAP values compared to the baseline value. Further investigation of quantization errors resulting in such poor performance of lower precision models is required. All the other models have comparative mAP values with the baseline TF model.

From the inference plot it can be concluded that TFLite low-precision models have very poor performance compared to the baseline. This could be attributed to TFLite quantized models not supporting NVIDIA GPUs and therefore running on the CPU. In contrast, TF-TRT models offer significant improvement in inference time compared to the baseline, as all the calculations are performed on the NVIDIA GPU.

The results from benchmarking TFLite's post-training quantized YOLOv4 version on Jetson Nano are shown in Figure 23. Like the Dell Workstation, the Int8 precision model has poor performance. This is in contrast to the claim in *TensorFlow Model Optimization* (n.d.), where the expectation is that Int8 precision would perform better on an ARM Neon processor. Hence, further investigation is warranted to understand the poor performance of low-precision models.

In general, the inference time for the baseline TF model is very low on Jetson Nano when compared to Workstation. The TF-TRT quantized models could not be run on Jetson Nano because of the large size of the model and low GPU memory.

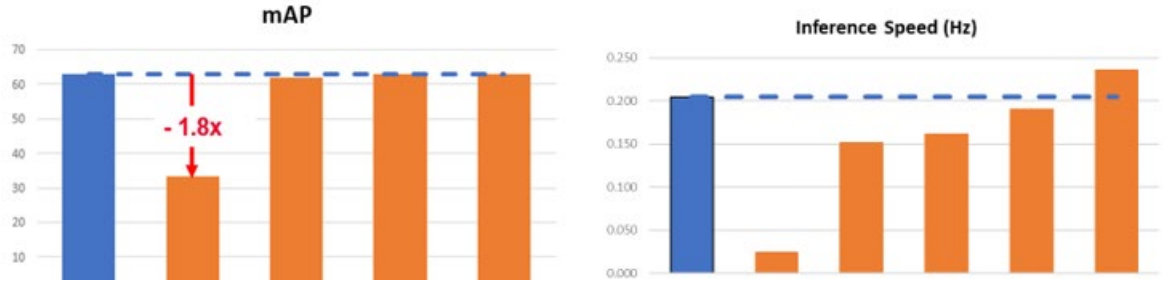

Figure 23. mAP and inference speed for different YOLOv4 quantized models on Jetson Nano

2.4.3 Summary

The summary of benchmarking of all the post-training quantized models of YOLOv4 is provided in Table 12. The red text is the baseline values of the column. All the percentages shown in blue are calculated with reference to the baseline values. Negative percentage values in mAP indicate degraded performance and indicate improved performance for the Inference Time column. It can be concluded that if high latency and accuracy is demanded from YOLOv4 and NVIDIA GPU is available, TF-TRT FP32 and FP16 YOLOv4 models would be an ideal choice.

Table 12. Benchmarking summary of YOLOv4 post-training quantized models

YOLOv4 TensorFlow Models		**Dell Workstation** **GPU:** NVIDIA GeForce GTX 1080, 8 GB **CPU:** Intel(R) Xeon(R) CPU E5-1660 v4 @ 3.20GHz, x86_64 **Memory:** 128 GB **OS:** Ubuntu 20.04.4 LTS Tensorflow v 2.9.1, 2.3			**Jetson Nano** **GPU:** NVIDIA Tegra, 128-core NVIDIA Maxwell™ architecture-based GPU **CPU:** Quad-core ARM® A57 **Memory:** 4 GB 64-bit LPDDR4; 25.6 gigabytes/second **OS:** Linux for Tegra®, Ubuntu 18.04 TensorFlow v2.7			**Size(MB)**
ML Library	**Quantization Method**	**COCO Dataset (1000 images)** **mAP(IOU=0.5)**	**Inference Time(sec)** **# threads : 4**	**Hz**	**COCO Dataset (1000 images)** **mAP(IOU=0.5)**	**Inference Time(sec)** **# threads : 4**	**Hz**	
TF (w/GPU)	*Baseline Original TF Model*	**61.3836593**	**0.038634**	**25.883**	**62.81831**	**4.886257**	**0.205**	**256**
TFLite (w/CPU only)	No Quantization/Float 32 *tf 2.3*	**61.34204 / -0.07%**	**1.05001 / 2617.83%**	**0.952**	**62.818 / -0.0005%**	**6.61694 / 35.4195%**	**0.151**	**257.6**
	Float 16 *tf 2.3*	**61.3364 / -0.08%**	**1.05016 / 2618.22%**	**0.952**	**62.8162 / -0.0034%**	**6.21037 / 27.0986%**	**0.161**	**129**
	Dynamic *tf 2.3*	**59.65175 / -2.82%**	**1.25655 / 3152.46%**	**0.796**	**61.8246 / -1.5819%**	**5.25711 / 7.5897%**	**0.190**	**64.7**
	Int8 *tf 2.9*	**32.48838 / -47.07%**	**1.5021 / 3788.02%**	**0.666**	**33.2942 / -46.9992%**	**4.24008 / -13.2244%**	**0.236**	**65.9**
	Int8 weights, int16 Activation *tf 2.9*	**-**	**12.541 / 32361.04%**	**0.080**	**-**	**40.941 / 737.8806%**	**0.024**	**66.1**
TensorRT (w/GPU)	No Quantization/Float 32 tf 2.3	**61.3837 / 0.0001%**	**0.01761 / -54.43%**	**56.795**				**1100**
	Float 16 *tf 2.3*	**61.3833 / -0.0006%**	**0.01757 / -54.52%**	**56.913**				**1100**
	Int8 *tf 2.3*	**38.19189 / -37.78%**	**0.01119 / -71.03%**	**89.346**				**597.6**

2.4.4 Conclusion

This section summarizes the effort to assess the performance of ML models using various optimization techniques. Various challenges were faced during the benchmarking process. Incompleteness of documentation related to incompatibility of various components of TensorFlow or TensorRT was a major challenge. The docker containers do not always work as-is for optimizations and require installation of appropriate versions of libraries to work.

The results from this report provide a deeper insight into the performance of an ML model when different post-training quantization optimizations are applied to it. Similar analysis could be performed to benchmark various other ML models. This type of benchmarking analysis assists in defining threshold criteria for selecting specific optimization technique when considering hardware deployment. Further, it could help guide the hardware requirements for an expected performance of ML model inference or vice versa.

In the near future, this analysis can be extended to include different performance assessment metrics, optimization techniques, hardware implementations (e.g. Tensor Processing Units), and ML frameworks such as PyTorch.

2.5 Uncertainty management in ML implementations

ML models, when implemented within the system pipeline, are essentially locally optimized to approximate functions, operating without a complete model of the world. As these learned models are transitioned into real-world deployment, decisions under uncertainty arise in relation to the model performance, safety, and security. To support these decisions, uncertainty must be quantified.

Quantifying the uncertainties inherent in the learned model becomes crucial in evaluating its performance when faced with unknown variables. By assessing and quantifying uncertainties, developers can better understand how the model behaves in the presence of unknown factors, thereby enhancing its reliability in real-world scenarios.

In the context of V&V, uncertainty quantification (UQ) plays a pivotal role in providing a measure of uncertainties stemming from various sources in real-world deployment. This quantitative assessment verifies the predictive capability of the model, offering insights into its reliability and accuracy when confronted with uncertainties.

Uncertainty management involves identifying, assessing, and mitigating potential risks resulting from unknown variables, as shown in Figure 24. The proposed approach enables informed

decision making, robustification of learned models, data fusion, safety, and assurance of the learned model. In many scenarios the output of a learned model will not be perfectly predicted, and the evidence that supports selection of one value over another is important. UQ serves as a critical layer of safety assurance that could lead to more principled decision making by enabling sound risk assessment and management.

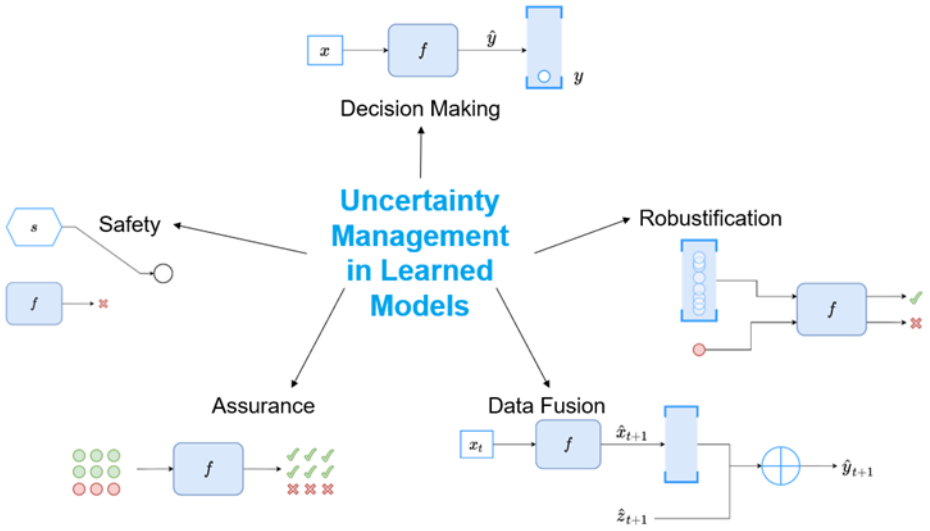

Figure 24. Various aspects of uncertainty management in learned models

2.5.1 Uncertainty in ML-based systems

Uncertainty is defined as lack of knowledge related to understanding of an event (DIN Media, 2024). To quantify and manage uncertainty about learned models, uncertainty about what must be considered. In selecting ML models as elements or components of a safety-critical system, there is uncertainty as to whether the inclusion of a ML model within the system will lead to system unsafe behaviors. This is a system-level type of question related to safety risk. A similar question can be also be considered at the component level. This section focuses on the component level, given the novelty in development process of ML components compared to traditional software. However, it is worth mentioning that a well-behaved component is necessary but not sufficient to ensure a well-behaved system, which would require a complete system approach.

At a conceptual level for model-based systems, the uncertainty consists of three qualitative dimensions: location of uncertainty, level of uncertainty, and nature of uncertainty (Walker, et al., 2003). The *location of uncertainty* dimension addresses narrowing down the source of uncertainty in the system pipeline. Generic locations for a system could include context, which

defines the boundaries of the system in ML models. It is usually represented by operational design domain (ODD), model uncertainty such as model structure uncertainty and technical uncertainty due to software or hardware errors, inputs to the model, parameter, and model outcome (Walker, et al., 2003).

The *level of uncertainty* is defined as a rating on the scale from where everything is known precisely to the *unknown unknowns*, or total ignorance. Between these extremes, level of uncertainty is broken down into *statistical uncertainty,* e.g. measurement uncertainty; *scenario uncertainty*, e.g. range in outcomes due to different underlying assumptions; and *recognized ignorance*, representing fundamental uncertainty about functional relationship (Walker, et al., 2003). Determining the level of uncertainty can be informed by (Smith, 2013):

- Uncertainty about the operational context
- Uncertainty about how the operational context will affect the real-world flight conditions
- Uncertainty about how the operational context will affect the sensors (input uncertainty)
- Uncertainty due to model inadequacy (structure and parameters, local/global properties)
- Uncertainty about model implementation (computational errors)

The *nature of uncertainty* assists in understanding how uncertainty can be addressed and how it arises in prediction. It is broadly categorized into two types: epistemic uncertainty and aleatoric uncertainty (Murphy K. P., 2022). *Epistemic uncertainty* is due to an imperfect understanding of the underlying causes or mechanism generating data and is also referred to as *model uncertainty*. *Aleatoric uncertainty* arises due to intrinsic variability and cannot be reduced even if more knowledge is acquired. Aleatoric uncertainty is also referred to as *data uncertainty*.

Often the aleatoric uncertainty is also described as *irreducible*. However, this does not always mean that epistemic uncertainty is reducible. To understand this, we need consider Claude Shannon's definition of information as *the resolution of uncertainty*, which implies that if information is available, epistemic uncertainty can be reduced (Jaoude, 2017). But this may not always be the case. An example of this may occur in parameter estimation, where parameter uncertainty cannot be reduced with the available data.

Classification of uncertainty types as described above is not universally accepted, because what may appear to be random in one state of knowledge may not be so in a different one. Therefore, many scientists consider that all uncertainty is epistemic in nature. Some scientists, like Jaynes (2003), even argue that ontological uncertainty is an example of a mind projection fallacy: we do

not understand nature's mechanisms at play, which appear to be random to us; and therefore, we attribute randomness to nature. From an epistemic perspective, the preferred explanation for variability in the results of repeated experiments is that the results are not reproducible by the experimental technique used due to factors we cannot control, and we do not know.

2.5.1.1 Uncertainty in learned models

The process of creating learned models generally involves data, algorithms, and computational resources. Learned models, often represented as deep neural networks with model parameters represented by weights and biases. The model structure is chosen by the modeler, including how many layers are in the neural network and how connected or sparse these layers are, i.e., how many non-zero weights. In some applications there may be also additional parameters called *hyperparameters*. Then, the weights are computed by iterating over data, which is referred to as the *training process*.

Uncertainty in learned models refers to confidence in the model output, and UQ describes the degree of uncertainty in the output. Some critical factors leading to uncertainty in learned models' predictions are as follows (Gawlikowski, J., Tassi, C.R.N., Ali, M. et al., 2023; Kläs, M., Vollmer, A.M., 2018):

- *Variability in real world situations*: When deployed in the real world, models could be affected by distribution shift in input, e.g. change in temperature, illumination.
- *Errors inherent to measurement systems*: Errors, e.g. sensor noise due to vibrations in the systems, contamination from bugs, occlusions for vision systems etc., can lead to poor quality of data used in training and calibration of learned models.
- *Approximation uncertainty*: There may be uncertainty in how well the model approximates the hypothesis, represented by true risk. DNNs are usually overparameterized, i.e. the number of parameters is greater than training data. As a result, model uncertainty (uncertainty about what is the right model to fit) essentially disappears, and approximate uncertainty still remains (Hüllermeier & Waegeman, 2021).
- *Omitted variables*: Some variables that were not included could offer relevant information, e.g. missing color in black and white images (Cornelia Gruber, Patrick Oliver Schenk, Malte Schierholz, Frauke Kreuter, Göran Kauermann, 2023). This leads to two key effects: an increase in prediction bias (systematic errors) and an increase in aleatoric uncertainty (random errors in prediction).

- *Errors caused due to missing and unknown data:* The lack of observed variables or data collection errors can lead to latent (unobserved) data that affects both input features and outputs. Sources of error, especially when left uncorrected, can result in biased models, leading to poor generalization in real-world scenarios.

2.5.1.2 Importance of UQ in learned models

Usually when learned models are assessed for their performance, they are assessed using individual predictions based on set thresholds or using measures on set samples. For example, Area Under Curve Receiver Operating Curve (AUROC) is typically used for performance measurement for classification problems at various threshold settings. A clear weakness in such methods is that performance is evaluated using set optimal thresholds on a test dataset. This performance will deteriorate under data distribution shift, and the set optimal thresholds will become invalid. This performance deterioration can be mitigated with UQ of the learned model.

UQ plays an important role for learned models used in various applications, such as engineering design, prognostics and health management, robotics, and computer vision in the automotive and aviation domains. Some examples are highlighted in this section.

In *Neural Network Based Runway Landing Guidance for General Aviation Autoland* (Balduzzi, et al., 2021), a neural network-based runway detection system using an ML model is proposed. The uncertainty information of runway key points is used to determine confidence in the model predictions. Case studies like Nemani, et al. (2023) investigate different UQ methods for ML models used for battery life prediction and turbofan remaining useful life prognosis. Various techniques such as neural network ensembles, Monte-Carlo Dropouts, Gaussian Process Regressions and Spectral-Normalized Gaussian Process are presented.

Choi, Castaneda, Tomlin, & Sreenath (2020) present a novel RL framework for safety-critical control in systems with model uncertainty, integrating CLF- and CBF-based QPs with RL. The framework, RL-CBF-CLF-QP, learns model uncertainties in dynamic constraints, enhancing safety and stability in uncertain environments.

Motion prediction models based on the LSTM are used for autonomous vehicles (Tang, et al., 2022). For quantifying uncertainty, the deep ensemble technique is applied. Deep ensemble estimates both epistemic and aleatoric uncertainty of the results predicted by the LSTM. The estimated uncertainty of motion prediction results is then transferred to the decision-making framework.

A brief and comparative survey of methods for UQ in DNNs, along with existing metrics to evaluate uncertainty predictions in the automotive domain, is presented by Arnez, Espinoza, Radermacher, & Terrier (2020).

Lastly, readers can also refer to the survey of various uncertainty toolkits for deep learning in (Pintz, Sicking, Poretschkin, & Akila, 2022) to gain information regarding various implementations.

2.5.1.3 A model uncertainty analysis approach

Any model uncertainty analysis approach seeks to quantify a hypothesis about the model quality. The formulation of the hypothesis depends on the application and type of ML.

The behavioral approach considers the model desired behavior $\mathcal{B}$ and seeks to test the hypothesis

$$H = \{the\ model\ behaviour\ \mathcal{B} \text{ will be exhibited over the model operational context}\}.$$

In the case of ML-reasoning models, the different model behaviors can be formulated depending on the model use: detection, classification, segmentation, clustering, or prediction of some continuous attribute. The behavioral approach is also applicable to RL where the model learned is a ML-agent model. In this case, the hypothesis is formulated in terms of the interaction between the model and its world environment.

The behavioral formulation goes hand in hand with the epistemic UQ approach, which can be used to compute the probability $P(H|D,I)$ that the hypothesis about behavior is true conditional on the verification data D and other relevant model background information I. Note that the probability $P(H|D,I)$ is quantifying uncertainty about the model behavior and not making statements that the behavior is a random variable. What is distributed is the probability, not the hypothesis nor the behavior.

The ontological approach can only associate probabilities to random variables. So, in this case, the UQ is done using null-hypothesis significance testing tools. This approach considers the model output error from the verification data

$$E = Y - \hat{Y}$$

where Y is the vector of labeled true outcomes and $\hat{Y}$ is the vector of model outputs based on the verification input data. The ontological approach, limited by considerations that probabilities can only be associated with random variables, computes the probability $P(E|H)$ using p-values, and from the third probability makes assertions about the hypothesis H being rejected at a significant test value.

The ontological approach substitutes the desired quantification, namely $P(H|E)$, by the quantification $P(E|H)$, based on p-values. The latter expresses how likely it is to obtain evidence E as extreme as that observed if the hypothesis H is true. Much of the modern literature in UQ advocates for stopping the use of null-hypothesis significance testing, since it not only substitutes the quantification we often want for another, but also has led to a replication crisis in science (Clayton, 2021).

Section 2.5.3.3 presents quantification of an ML classifier, an ML detector, and an ML predictor using the behavioral approach with epistemic UQ.

2.5.2 Uncertainty management in learned models

To effectively handle the uncertainty associated with systems using learned models, several considerations have been proposed, as illustrated in Figure 25. These considerations span across different phases in the lifecycle of the system containing the learned model, including design, development, deployment, and operational phases. While the list of considerations is not exhaustive, it offers an initial approach. By incorporating these considerations, practitioners can work towards reducing uncertainty and improving the reliability of the learned models.

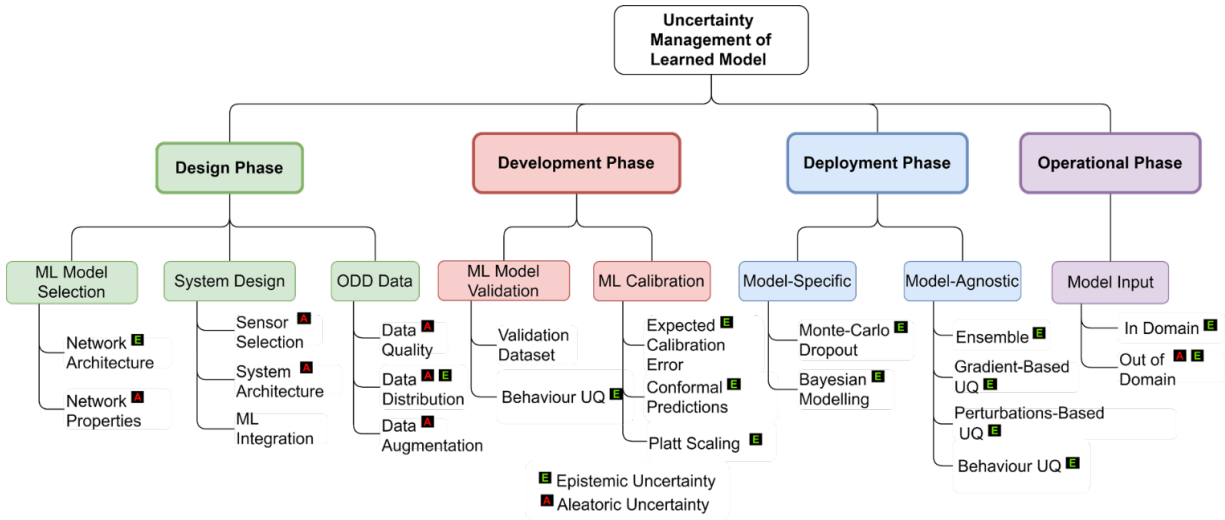

Figure 25. Methods for uncertainty management during different design phases of learned model

2.5.2.1 Design phase

During the design phase, careful attention must be given to the selection of appropriate algorithms and model architectures. This involves evaluating the trade-offs between model

complexity and interpretability, as well as considering the availability and quality of training data.

2.5.2.1.1 Learned model selection

An important aspect during the design phase is the appropriate selection of a learning algorithm, architecture, and model. This is partly driven by the type of task, such as classification, regression, transcription, machine translation, anomaly detection, synthesis sampling, imputation of a missing value, denoising, etc. The model architecture specifies the function and initializes parameters which the learning algorithm varies to reduce the training objective. Explicit guidance on defining the neural network, i.e. its depth and width, does not exist, and ideal network architecture is found by experimentation guided by validation set error. However, some starting points on architecture selection can be found in *Neural Network Design* (Hagan, Demuth, Beale, & De Jesus, 2014). However, specific to neural networks models, the Universal Approximation Theorem states that for large neural networks, regardless of the function, their attempts to learn will be able to represent the function (Goodfellow, Bengio, & Courville, Ch 6. Deep Forward Networks, 2016).

Learned model selection is usually based on predictive and generalization performance on data (Raschka, 2020). Further, while selecting a model, it is often challenging to strike a balance between underfitting (high bias) or overfitting (high variance) the data, which could lead to poor generalization. Such challenges can be addressed by using UQ to enhance model selection by incorporating uncertainty metrics alongside traditional accuracy-based criteria (Barandas, Folgado, Santos, Simão, & Gamboa, 2022). UQ helps in choosing models that are not only accurate but also more reliable and robust. By focusing on models that generate low uncertainty and have predictable behavior, UQ enhances the selection of models suited for tasks with safety, reliability, or interpretability concerns.

2.5.2.1.2 System design

Designing a system with a learned model as one component is a preferred approach compared to a complete end-to-end learned model for safety-critical systems. System designs with a learned model as a component decomposes the pipeline into different functions and makes it relatively easier to debug/assess compared to end-to-end learned model systems.

A generic pipeline for computer vision system would include electro-optical sensor, detector, tracker, planner, and a decision-making module (Kakaletsis, et al., 2022). Selection of each of the modules is application-based and critical for managing uncertainty. A variety of homogenous or heterogenous sensors suite can be selected, and they play a crucial role in uncertainty

management. Examples of sensors include cameras (e.g., thermal imaging cameras, RGB cameras), radar, light detection and ranging (LIDAR) (United States Patent No. US10878709B2, 2020; Sanders-Reed, 2021). Szeliski (2022) provides a comprehensive introduction to readers interested in learning more about digital cameras. An interesting approach to ML-enabled sensor design is proposed by Ballard, Brown, Madni, & Ozcan (2021), namely to influence hardware decision from the outset. Novel aspects include optimizing the sensor system using a cost function and integrating features through iterative feedback loops, rethinking traditional sequential methods of sensor hardware and software.

Detectors are usually based on convolution neural network models (Lecture 9: Object Detection and Image Segmentation, 2024). Planner and decision-making modules can be based on logic or heuristics (United States Patent No. 11,908,336, 2024; LaValle, 2006). Lastly, for trackers, Kalman filters are a usually a popular choice, providing the covariance and mean information of the state being tracked.

2.5.2.1.3 Data content

The ODD in the context of ML, particularly in autonomous systems, refers to the specific conditions under which an ML model or autonomous system is designed to operate safely and effectively. These conditions include geographic area, environmental conditions (like weather and lighting), road types, traffic scenarios, etc. In autonomous systems, for example, the ODD defines the boundaries of operation, such as whether the vehicle can function in urban or rural environments, during daylight or nighttime, in snowy or rainy weather, etc. By explicitly defining the ODD, developers ensure that the ML models or systems only function within their validated operational scope, minimizing risks and improving safety.

Data distribution used in the ML lifecycle must be based on ODD. Important quality characteristics of this data include coverage of labelling accuracy, consistency, and absence of annotation bias. It is critical to define data quality requirements, as labelers can differ in their level of uncertainty or reliability and their central tendency while labelling, which carries forward to data. Cappi, et al. (2021) provide some recommendations for improving data quality. Also, heterogeneity of labelers violates the “independent and identically distributed” assumption. If this assumption holds for the model, the aleatoric uncertainty can be reduced (Gruber, Schenk, Shierholz, Kreuter, & Kauermann, 2023).

Data quality can also be improved by detecting data anomalies in learned model pipelines, where input data errors can degrade model performance and cause operational issues. The anomalies could be missing values, data skew, or drift. Polyzotis, Zinkevich, Roy, Breck, & Whang (2019)

proposes a schema-based validation system to detect anomalies in ML pipelines. It validates input data by comparing it to a predefined schema that encodes expected properties like data types, ranges, and feature presence. The system detects anomalies both within single batches of data (e.g., missing values or incorrect types) and across batches (e.g., data drift or distribution skew).

Data augmentation is the process of modifying the existing data to create additional training datasets. It involves making minor changes to original data to increase diversity, size of training set, etc. Jiang, Zhao, Wu, & Zuo (2022) propose a novel approach to capture model uncertainty in neural networks using data augmentation techniques like adding Gaussian noise. The key insight is that augmenting input data can be mathematically equivalent to applying transformations on model weights, which approximates Bayesian inference. By introducing data augmentation during both training and inference, the method generates multiple predictions for each input, simulating different realizations of the model. This allows the estimation of predictive uncertainty without requiring the computational expense of Bayesian neural networks.

2.5.2.2 Development phase

In the development phase, it is essential to thoroughly validate the learned model using appropriate evaluation metrics and cross-validation techniques. This helps to assess the model's performance and identify potential sources of uncertainty. Sensitivity analysis can also be conducted alongside performance evaluation to understand the impact of input variations on the model's predictions, providing insights into its robustness. Lastly, it is also important to calibrate the learned models to provide reliable uncertainty information.

2.5.2.2.1 Learned model validation

Learned model validation is an important step in the development process. Model validation is crucial because it ensures that an ML model is not only performing well on the training data but can also generalize effectively to unseen data. This is particularly important in real-world applications like aviation, where overfitting to the training data can lead to inaccurate predictions on new data, thereby compromising decision-making. Proper validation helps assess the model’s reliability, robustness, and ability to generalize across different datasets and populations.

Key methods for model validation include Holdout Validation, which splits data into training, validation, and test sets; and Cross-Validation, where techniques like K-fold and Leave-One-Out split data iteratively to provide more reliable performance estimates (Maleki, Farhad et al, 2020). Nested Cross-Validation is used for hyperparameter tuning, ensuring an unbiased generalization error estimate, though it is computationally intensive. Leave-One-Group-Out Cross-Validation is

applied when data points are related (e.g., multiple images from the same class), preventing data leakage. External validation, where models are tested on data from different institutions, is emphasized for demonstrating real-world generalization. In cases of small datasets, data augmentation and patch-based approaches can increase sample sizes, but care is needed to avoid overfitting.

2.5.2.2.2 Learned model calibration

Learned model calibration is the process of matching the predicted probabilities of a model to the empirical frequencies or true correctness likelihood. Calibration is important for safety-critical systems, as accurate and reliable predictions will lead to informed decision making. However, learned models are not guaranteed to be well calibrated post training and can therefore be miscalibrated (Johansson & Gabrielsson, 2019). Elements responsible for miscalibration (Guo, Pleiss, Sun, & Winberger, 2017) include *model capacity*, represented as depth and width; *batch normalization*, which can improve the training but also affects final model prediction; and less regularization via *weight decay*.

Various metrics such as expected calibration error, maximum calibration error, pinball loss, brier score, quantile calibration error, negative log likelihood, etc. can be used to measure calibration error (DIN Media, 2024; Guo, Pleiss, Sun, & Winberger, 2017). They are commonly visualized by plotting a reliability diagram. Also, there various post-hoc calibration methods such as Platt Scaling, histogram binning, isotonic regression, vector scaling, matrix scaling, conformal prediction, etc.

Nixon, et al. (2020) have conducted a comprehensive empirical study evaluating various calibration measures. A comparison of Static Calibration Error (SCE) and Adaptive Calibration Error (ACE) and Expected Calibration Error (ECE) for Wide ResNet and Multinomial Logistic Regression models trained on CIFAR-10 and MNIST, respectively. The paper recommends using class-conditional calibration and adaptive binning for more accurate measurement and suggests using the L2 norm over the L1 norm for better calibration optimization. It also advises evaluating all predictions (not just the maximum probability) and applying thresholding to ignore insignificant probabilities in multiclass settings. These changes address shortcomings in the widely used ECE metric.

Two methods, vector scaling and conformal prediction, will be discussed, as they were used in the case study presented in this report. An additional list of calibration implementations can be found in *Implementations of Uncertainty Quantification Methods for Neural Networks* (2021).

2.5.2.2.3 Vector and matrix scaling

Vector and matrix scaling is a calibration approach for multilabel classification problems which have more than two class labels (Guo, Pleiss, Sun, & Winberger, 2017). This is an extension of Platt scaling (Platt, 1999), where a matrix is used to scale the logits (pre-softmax outputs) for each class.

In vector scaling, a vector is learned that applies a class-specific scaling to the logits before computing the softmax probabilities. This results in each class's confidence being adjusted independently and the optimization of the model's scaling parameters using the negative log-likelihood loss on the validation set.

A more general extension, matrix scaling applies a full linear transformation to the logits by learning both a weight matrix and a bias vector. The logits are multiplied by this matrix, and a bias is added before the softmax operation. Matrix scaling is more expressive than vector scaling, as it allows for inter-class dependencies to be modeled. However, this also introduces many more parameters, making it prone to overfitting, especially when there are many classes.

In an uncalibrated neural network model with logits vector z_i, the prediction score is obtained using a maximum class score from the softmax function shown below:

Softmax Function, $\sigma_{SM}(z_i)^{(k)} = \frac{\exp\left(z_i^{(k)}\right)}{\Sigma_{j=1}^{K} \exp\,(z_i^{(j)})}$, confidence score $\hat{p}_i = \max_k \sigma_{SM}(z_i)^{(k)}$

Class prediction $\hat{y}_i = argmax_k\, z_i^{(k)}$

The application of linear transformation to logits $Wz_i + b$

Calibrated confidence score $\widehat{q_i} = \max_k \sigma_{SM}(Wz_i + b)^{(k)}$,

Calibrated class prediction $\widehat{y_i'} = argmax_k\, (Wz_i + b)^{(k)}$

2.5.2.2.4 Conformal predictions

Conformal prediction is a UQ framework for ML to produce probabilistic predictions. These predictions can be sets for a classification problem or bounds for a regression problem. The framework offers strong mathematical assurance that the error rate will never surpass the significance level determined by the user. Some advantages of using conformal prediction for UQ include (Manokhim, Valery, 2023):

- *Probabilistic prediction:* Provides probabilistic model predictions which include confidence, accuracy, and reliability for each prediction.

- *Coverage guarantees:* Provides rigorous statistical validity guarantees.
- *Non-parametric:* Does not require assumption of underlying probability distribution.
- *Distribution agnostic:* Prediction models work for any data distribution as long as exchangeability assumptions can be maintained.
- *Robustness:* Robust to outliers and noisy data.
- *Low Computation Cost:* Requires low computational resources to get probabilistic predictions.

The basic components of conformal prediction are a nonconformity score and calibration set. The *nonconformity score* is a function that measures the difference between a new data point and existing data points. The nonconformity scores are broadly categorized as model-dependent and model-agnostic. Selection of nonconformity score is critical, as it substantially effects the prediction sets (Johansson, Linusson, Löfström, & Boström, 2017; Aleksandrova & Chertov, 2021). Some examples of nonconformity scores include Hinge Loss, Margin Conformity Measure, and Brier Score. The *calibration dataset* is a portion of the dataset used to calculate nonconformity scores for the known data points. This assumes that the new test data point comes from the same distribution as the calibration dataset and raises concerns if the new data point is not from the same distribution. To handle such scenarios, extensions of conformal predictions are also available in literature which can handle covariate or data distribution shifts (Angelopoulos & Bates, 2022).

The workflow of conformal prediction is shown in Figure 26. It consists of the following steps (Manokhim, 2023):

1. A conformal predictor learns from the calibration dataset to quantify uncertainty around predictions for new observations.
2. To quantify uncertainty around a new observation, the predictor calculates the non-conformity score and uses it to determine if the new observations fall within the range of expected values.
3. The model then calculates confidence measures and prediction score or intervals for each prediction.

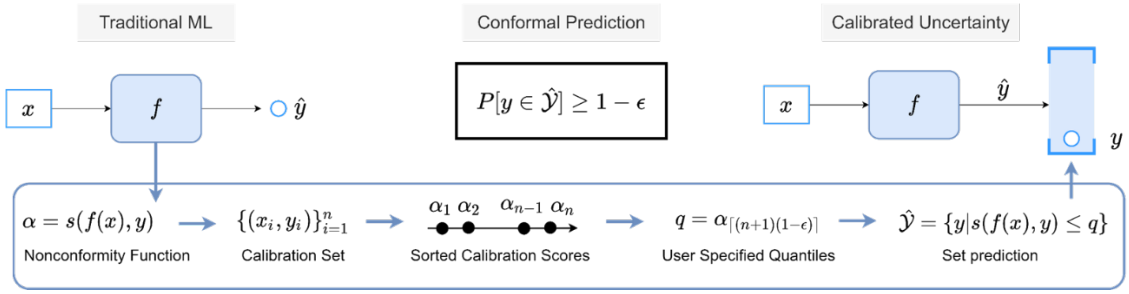

Figure 26. Workflow for conformal prediction of a generic ML model

2.5.2.3 Deployment phase

During the deployment phase, considerations should be given to the model's integration with the target system or application. This involves addressing issues related to data compatibility, model versioning, and monitoring. Various model-agnostic and model-specific implementations are also considered during deployment phase. Model-agnostic techniques include methods such as *model ensembling*, or deploying multiple models to reduce the risk of relying on a single model's predictions. This can enhance the model's reliability and provide more robust and accurate results in real-world scenarios. Model-specific techniques include Bayesian neural networks (BNNs), GPR, etc. Interested readers can also see Appendix A.

2.5.2.3.1 Model-specific methods

BNNs are a vital method for capturing uncertainty in DNNs by modeling the network's parameters as probability distributions rather than fixed values. This enables BNNs to address both epistemic uncertainty (model uncertainty due to a lack of data) and aleatoric uncertainty (inherent data randomness). The Bayesian approach places a prior distribution over the network weights and uses inference to obtain a posterior distribution, allowing predictions to effectively integrate uncertainty.

BNNs employ various techniques to approximate the intractable posterior distribution of model parameters. One key method is *variational inference* (Blei, Kucukelbir, & McAuliffe, 2017)**,** which optimizes over a family of simpler distributions to make posterior approximation computationally feasible for large models. *Sampling approaches***,** such as Markov Chain Monte Carlo (MCMC) and its variant, Hamiltonian Monte Carlo (HMC), provide a more direct but computationally expensive method by sampling from the true posterior. Lastly, the *Laplace approximation* (Kristiadi, Hein, & Hennig, 2021) fits a Gaussian distribution around the mode of the posterior, offering a simpler and efficient approximation. This is especially useful for pre-

trained models. Each technique offers a trade-off between computational cost and approximation accuracy.

2.5.2.3.2 Model-agnostic methods

This section provides an overview of various model-agnostic methods. Shen, et al. (2023) introduce a Bayesian meta-model using a Dirichlet distribution to enhance post-hoc UQ in neural networks. By leveraging features from different model layers, it captures both epistemic and aleatoric uncertainties, improving performance on tasks like out-of-distribution detection and misclassification without requiring model retraining, making it computationally efficient. Methods for pre-trained networks have also been proposed for UQ (Wang & Ji, 2024). These approaches estimate uncertainty by either applying perturbations to the model's weights or by analyzing the gradients of the parameters relative to a given input. Ensembles are a popular approach and discussed in detail, as they are used in the case study.

Ensemble learning, also known as ensemble modeling, combines predictions from multiple models to make a final prediction. When it comes to deep learning models, which require significant computation and data, ensemble deep learning models require special consideration. These models must effectively incorporate complementary information from multiple algorithms into a unified framework. Challenges for ensemble deep learning models include ensuring diversity among baseline models, minimizing training time and model complexity for practical applications, and effectively merging predictions from complementary algorithms.

There exist various strategies for creating ensemble models (Ganaie, Hu, Malik, Tanveer, & Suganthan, 2022; Ensemble-PyTorch Developers, 2021):

- *Fusion* averages output from all base estimators. During the training stage, all base estimators in fusion are jointly trained with mini-batch gradient descent.
- *Voting and bagging* fit base estimators independently, and the final prediction takes the average over the predictions from all base estimators. The main idea in bagging is to generate a series of independent observations with the same size and distribution as that of the original data. Given the series of observations, they generate an ensemble predictor which is better than the single predictor generated from the original data.
- *Gradient boosting* trains all base estimators in a sequential fashion, as the learning target of a base estimator is associated with the outputs from base estimators fitted before.
- *Fast geometric ensemble* uses a customized learning rate scheduler to generate base estimators.

- *Negative correlation-based deep ensemble methods* encourage diversity among the individual models of the ensemble to learn the diverse aspects of the training data. These methods minimize the empirical risk function of the ensemble model via minimization of error functions of the individual networks.
- *Explicit/implicit ensembles* are a single model trained to behave like an ensemble, training multiple neural networks without incurring additional cost or keeping the additional cost as minimum as possible. Examples include DropOut and DropConnect.
- *Heterogenous/homogenous ensembles* train a group of base learners either from the same family or different families. The base learner can be a decision tree, neural network, or any other learning model.

A popular approach as proposed by Lakshminarayanan, Pritzel, & Blundell (2017) offers a simple, scalable, and effective deep ensemble of neural networks for predictive uncertainty estimation, offering results as good as or better than approximate BNNs while being easier to implement and computationally efficient. Novel aspects include using proper scoring rules (like negative log likelihood) as training objectives, employing adversarial training to smooth predictive distributions, and avoiding the computational complexity of Bayesian methods. The ensemble is treated as a uniformly weighted mixture mode and combines the predictions as follows:

$$p(y|x) = \frac{1}{K}\sum_{i=1}^{K} p(y|x, \theta_k)$$, where K is total number of estimators.

This prediction for classification provides average predicted probability, and for regression, a mixture of Gaussian distribution.

2.5.2.4 Operational phase

In the operational phase, continuous monitoring and evaluation of the learned model's performance are crucial. This includes tracking the model's performance and detecting concept drift.

2.5.2.4.1 Model input

Uncertainty management in the operational phase of an ML model involves identifying both in-domain (ID) and out-of-domain (OOD) inputs to ensure reliable performance. OOD detection mitigates overconfidence in neural networks, which tend to produce high-confidence predictions even on unfamiliar or OOD data.

For in-domain data, uncertainty is quantified using techniques like BNNs, Monte Carlo Dropout, and ensemble methods to generate confidence intervals and improve model robustness, as mentioned previously. Probability calibration and monitoring for data drift further help maintain model accuracy. An approach to monitor data drift proposed by Ackerman, Raz, Zalmanovici, & Zlotnick (2021) is called the *classifier confidence auditor*, which detects data drift by monitoring changes in the classifier's confidence on predicted labels without requiring labeled production data.

For detecting OOD data, Yang, Zhou, Li, & Liu (Yang, Zhou, Li, & Liu, Generalized Out-of-Distribution Detection: A Survey, 2024) provide methods in four categories, namely classification-based, density-based, distance-based, and reconstruction-based methods. These strategies are essential for managing uncertainty and ensuring the model's robustness in real-world applications.

Classification methods include *energy-based* methods for OOD detection. These methods calculate the energy score, which can be interpreted from a likelihood perspective, to differentiate between ID and OOD samples. Specifically, energy-based models (EBMs) assign a scalar energy score to inputs, with lower energy values indicating ID samples and higher energy values flagging OOD samples.

Density-based methods for OOD detection explicitly model the in-distribution data and flag samples from low-density regions as OOD. Some examples include class-conditional Gaussian distribution, flow-based methods, likelihood ratio methods, etc.

Distance-based methods rely on the notion that OOD samples should lie far from the centroids or prototypes of ID classes. Some examples include Mahalanobis distance, cosine similarity, nearest neighbor distance, etc.

Lastly, *reconstruction* methods for OOD detection are based on an encoder decoder framework on ID data and OOD samples, and difference in model parameters can be utilized as an indicator.

2.5.3 Case study

In this section, we present a case study for an image classification problem and its performance degradation with respect to different data distributions. UQ and calibration methods are used in this problem to gain insights into their workflow. The goal of the study is a walkthrough of the ML development process with steps including data selection categories: ID, distribution shift, OOD, model selection and training, model calibrating, and quantifying output prediction uncertainty. It is also acknowledged that the case study provides an overview of a particular UQ

and calibration method and is not a comprehensive evaluation of the techniques. This study can be further augmented by benchmarking it against different UQ methods.

2.5.3.1 Problem definition

The problem is to design a multi-class image classifier. In this problem, labelled data is available, and the goal is to predict a known class label, making a supervised learning approach appropriate. The task is to learn a mapping *f* from input $x \in X$ to outputs $y \in Y$. In this case, the input space is an image, and the output space is a mutually exclusive class label.

2.5.3.1.1 Data description

The Fashion MNIST dataset is used as the in-distribution dataset (Xiao, Rasul, & Vollgraf, 2017). This is a popular dataset used for image classification tasks. It consists of 70,000 grayscale images of fashion-related items, such as clothing, shoes, and accessories. Each image is 28x28 pixels, and the dataset includes 10 different classes, such as T-shirts, trousers, pullovers, dresses, and sneakers. It serves as a more challenging alternative to the original MNIST dataset, which contains handwritten digits, while maintaining the same structure and size. Fashion MNIST is often used to benchmark ML algorithms and to practice deep learning techniques on real-world data. Distribution shift in the Fashion MNIST dataset is obtained by applying geometric distortions. A portion of the dataset was also held separately for calibration.

To assess how the model handles OOD data, the MNIST dataset is used (Lecun, Cortes, & Burges, n.d.). The MNIST dataset is a widely used benchmark in the field of MLand computer vision. It consists of 70,000 grayscale images of handwritten digits. Each image is 28x28 pixels, with digits ranging from 0 to 9, making a total of 10 classes. The dataset is split into 60,000 training images and 10,000 testing images. MNIST is often used for image classification tasks and is considered a simple introductory dataset for deep learning, helping researchers and developers test algorithms and neural network architectures due to its simplicity and good labelling. See Figure 27 for examples of data from the Fashion MNIST and MNIST dataset.

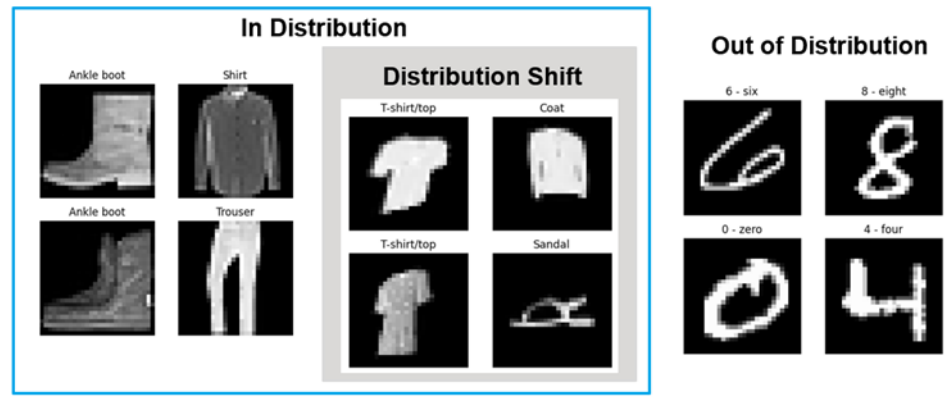

Figure 27. Data used for case study

2.5.3.1.2 Learned model description

A TinyVGG model is a simplified version of the VGG architecture and often used for quick experimentation on smaller datasets. This is referred to as the *baseline model*. It is a CNN built using PyTorch. It includes two blocks of convolutional layers followed by pooling and activation functions. These layers aim to capture hierarchical features in images, from low-level patterns like edges to high-level representations like object outlines. The details of the layers and shape are shown in Table 13.

Table 13. TinyVGG model summary

Layer (type (var_name))	Input Shape	Output Shape	Param #	Trainable
FashionMNIST_Baseline (FashionMNIST_Baseline)	[1, 1, 28, 28]	[1, 10]	--	True
├─Sequential (layer_block1)	[1, 1, 28, 28]	[1, 10, 14, 14]	--	True
│ └─Conv2d (0)	[1, 1, 28, 28]	[1, 10, 28, 28]	100	True
│ └─ReLU (1)	[1, 10, 28, 28]	[1, 10, 28, 28]	--	--
│ └─Conv2d (2)	[1, 10, 28, 28]	[1, 10, 28, 28]	910	True
│ └─ReLU (3)	[1, 10, 28, 28]	[1, 10, 28, 28]	--	--
│ └─MaxPool2d (4)	[1, 10, 28, 28]	[1, 10, 14, 14]	--	--
├─Sequential (layer_block2)	[1, 10, 14, 14]	[1, 10, 7, 7]	--	True
│ └─Conv2d (0)	[1, 10, 14, 14]	[1, 10, 14, 14]	910	True
│ └─ReLU (1)	[1, 10, 14, 14]	[1, 10, 14, 14]	--	--
│ └─Conv2d (2)	[1, 10, 14, 14]	[1, 10, 14, 14]	910	True
│ └─ReLU (3)	[1, 10, 14, 14]	[1, 10, 14, 14]	--	--
│ └─MaxPool2d (4)	[1, 10, 14, 14]	[1, 10, 7, 7]	--	--
├─Sequential (logit_layer)	[1, 10, 7, 7]	[1, 10]	--	True
│ └─Flatten (0)	[1, 10, 7, 7]	[1, 490]	--	--
│ └─Linear (1)	[1, 490]	[1, 10]	4,910	True
Total params: 7,740				
Trainable params: 7,740				
Non-trainable params: 0				
Total mult-adds (M): 1.15				
Input size (MB): 0.00				
Forward/backward pass size (MB): 0.16				
Params size (MB): 0.03				
Estimated Total Size (MB): 0.19				

The model is trained by multiple passes over the data. Over several epochs, the model's accuracy improves as it learns from its misclassification. Each epoch represents a full pass over the entire

training dataset. Initially, the model shows a training accuracy of 76.87, but after three epochs, it reaches 88.60, showing the model's ability to learn the relationships within the data. The accuracy of the model on test data was observed to be 87.61.

2.5.3.2 Experimental results

The performance of the model is assessed on the ID dataset using visualizations and summaries as obtained from plotting the confusion matrix, as shown in Figure 28. The confusion matrix provides valuable insight into the model's performance by showing where the model makes correct and incorrect predictions for each class. It helps assess the model's strengths and weaknesses by highlighting *diagonal entries*, which represent correct classifications. High values on the diagonal indicate that the model predicts certain classes well. *Off-diagonal entries* represent misclassifications, showing which classes are commonly confused with others. For example, the model may confuse similar-looking items, like T-shirts, shirts and coats, revealing areas where the model struggles.

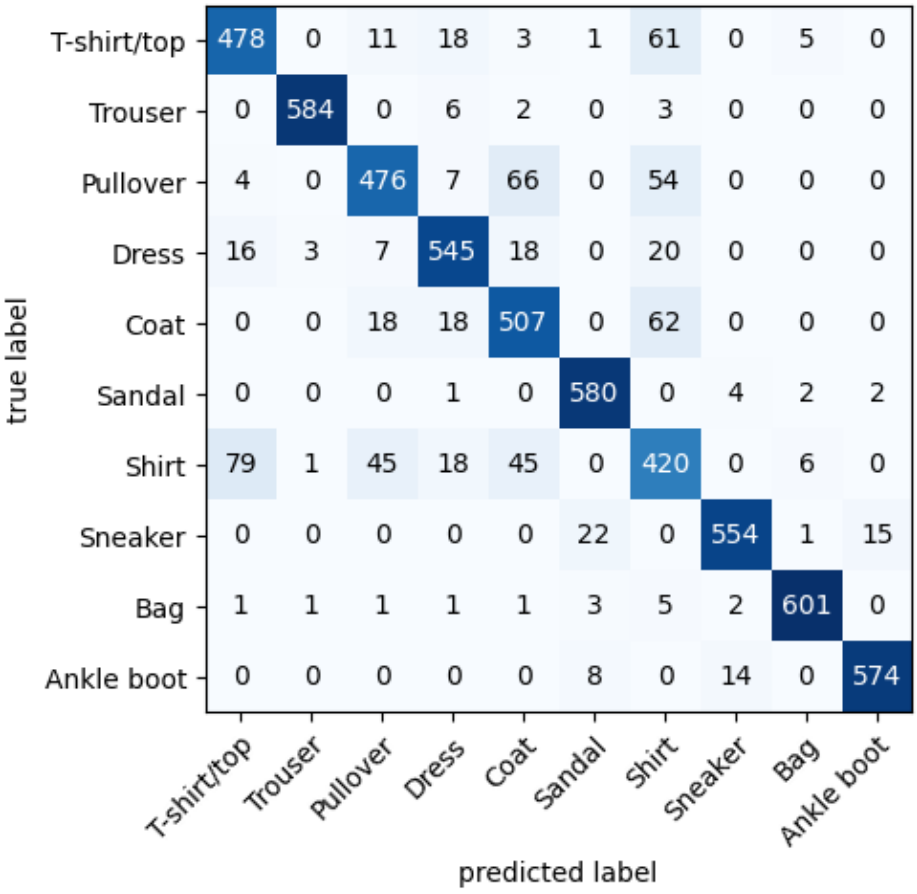

Figure 28. Confusion matrix of TinyVGG trained on Fashion MNIST dataset

Some examples were also visualized to understand the model's behavior, as shown in Figure 29. As seen in the top right corner, the model provides a high score for classifying coat as shirt. This example correlates with the confusion matrix.

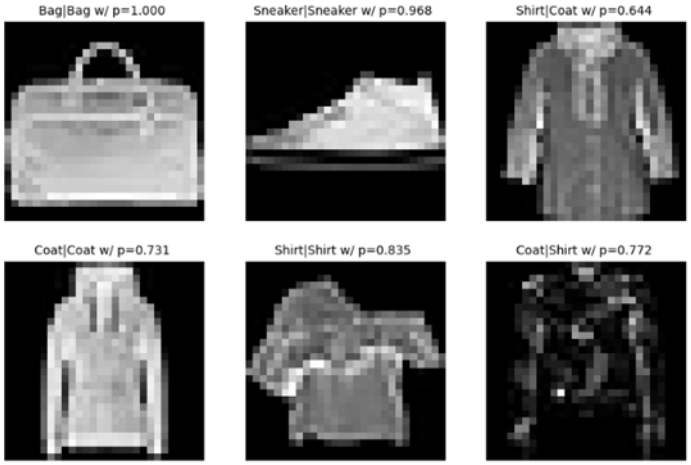

Figure 29. Prediction score from baseline model, Prediction | True Label with p = score

The accuracy of the baseline model was also assessed on transformed data and was observed to degrade significantly to 37.45.

2.5.3.2.1 Calibration of learned model

Assessment of the model calibration is conducted post-performance assessment. Vector scaling is applied to adjust prediction probabilities and improve model calibration. Calibration is essential to ensure that confidence in predictions (probabilities) aligns with actual likelihood of correctness. For instance, a well calibrated model should give a prediction with 70% confidence if it is correct roughly 70% of the time. Calibration improves reliability of the model, particularly in applications where uncertainty plays a key role.

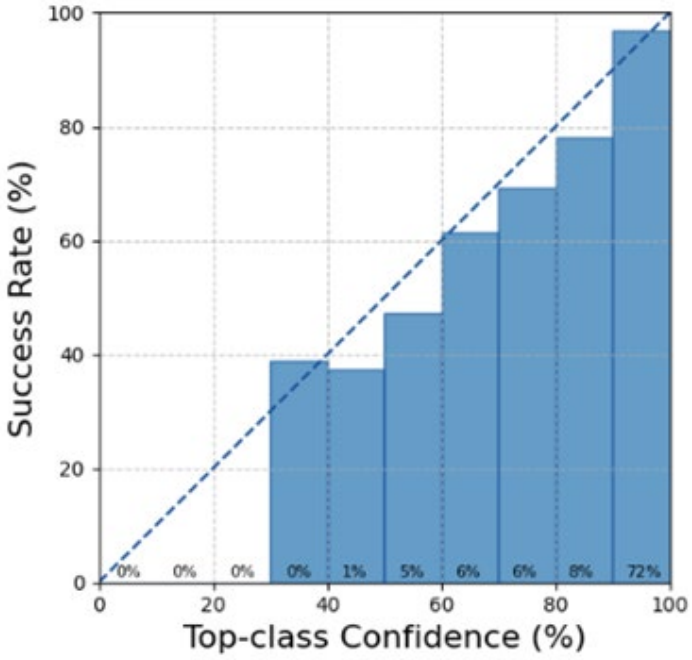

Figure 30. Reliability diagram for baseline model

At first, the reliability diagram is plotted for the baseline model, shown in Figure 30. It is observed that the model is slightly overconfident for higher values and underconfident in lower values. A perfectly calibrated model would track along the dashed line with a slope of 1. The miscalibration is calculated using an ECE metric, which was found to be 0.87. An ideal ECE score would be 0. An experiment was conducted to apply vector scaling to calibrate the model. However, it was observed that post vector scaling, the ECE value increased to 1.04. This needs further investigation. Some potential reasons for this increase include (Ding, Liu, Jinjun, & Shi, 2019; Nixon, et al., 2020):

- Calibration data is not representative of the test data. The model may become overly confident or under-confident on new, unseen data, leading to an increase in the ECE.
- If there is a mismatch between the distribution of the calibration set and the test set (e.g., due to distribution shift or presence of OOD data), the calibration can degrade performance on the test set.

2.5.3.2.2 Ensemble

To obtain uncertainty estimates and improve classification performance, deep ensemble technique is applied. An ensemble of TinyVGG models is trained, providing a more robust prediction by aggregating outputs from multiple models. The ensemble not only improves

accuracy but also helps in better quantifying uncertainty, as disagreements among ensemble members indicate high uncertainty. For this case study, an ensemble of three estimators was selected using a voting-based ensemble method. Each base estimator is trained independently, and the final prediction takes the average over predictions from all base estimators.

Table 14. Comparison between baseline model and voting ensemble

	Baseline TinyVGG	**Voting Ensemble**
Accuracy on test data	87.61	89.18
Accuracy on transformed data	37.45	50.71

The performance of the ensemble was assessed on test and transformed data, as shown in Table 14. It can be observed that the voting ensemble performs incrementally better for ID test data and significantly better on transformed data compared to the baseline model.

Figure 31 provides visualization of some of the results. It is observed that when the ensemble is very confident in its prediction with a high score the standard deviation is small. However, in some cases, when the ensemble has low confidence in its prediction, the standard deviation can be small as well. Similar observations were made for transformed data, as shown in Figure 32.

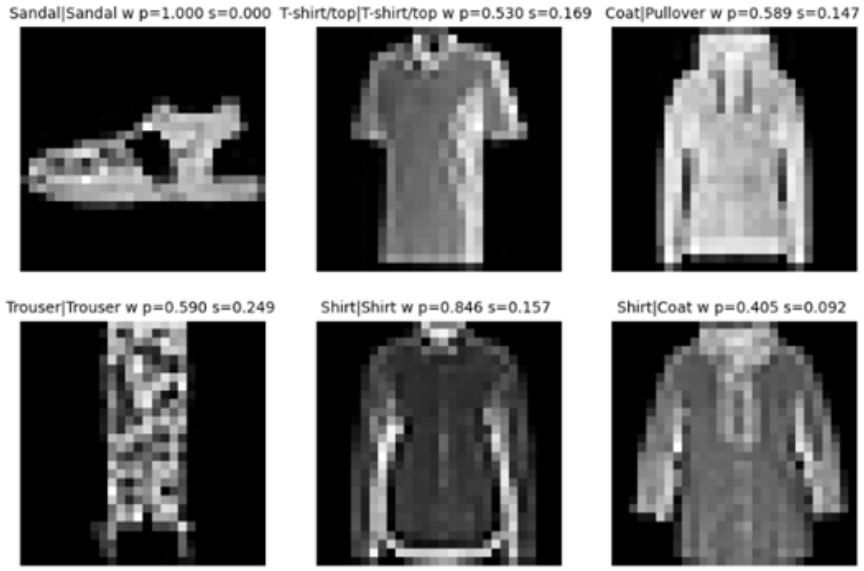

Figure 31. Visualization of ensemble predictions for in-distribution data, Predicted Label| True Label with p=mean score, s = std

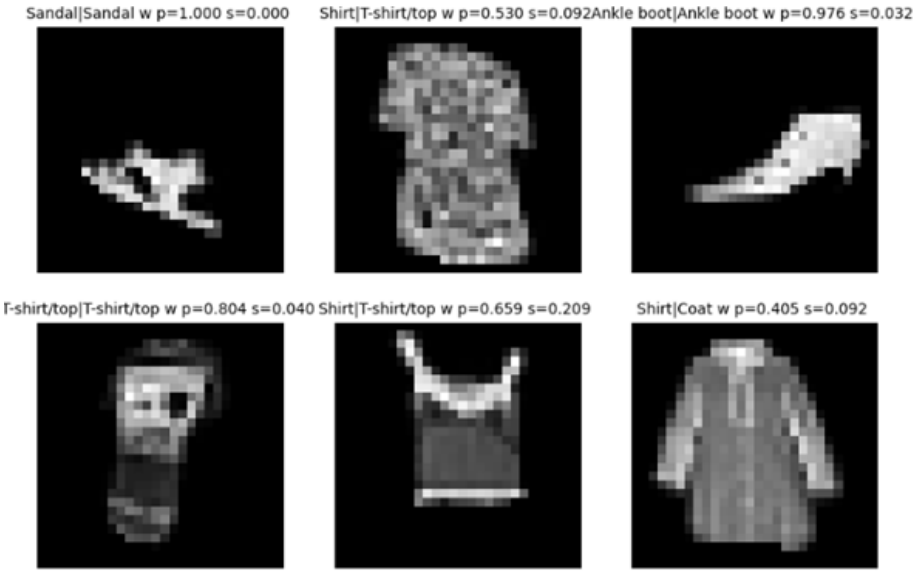

Figure 32. Visualization of ensemble predictions for distribution-shift data, Predicted Label| True Label with p=mean score, s = std

A TinyVGG Ensemble was also assessed for its capability to detect OOD data in the original MNIST dataset. Figure 33 shows the mean prediction score and its uncertainty estimates. It is interesting to observe that in the current implementation choice, the uncertainty of ensemble is very poor in the sense that it is overconfident in its predictions. Deep ensembles combine the predictions of multiple models, which can reduce the uncertainty in ID data. However, for OOD data, this combination often leads to highly confident, incorrect predictions. The ensemble members tend to give confident outputs on unfamiliar data, masking the fact that the input is far from the training distributions (Fang, Tao, Huang, & Yang, 2024). Also, as ensembles are trained on the same data, this lack of diversity diminishes the ability of the ensemble to detect when data is OOD (Xia & Bouganis, 2022). An ensemble method such as the one presented by Yangf,f Ngoc, Shin, Lee, & Hwang (2021) could help improve ensemble OOD detection, but it needs further investigation and empirical evaluation.

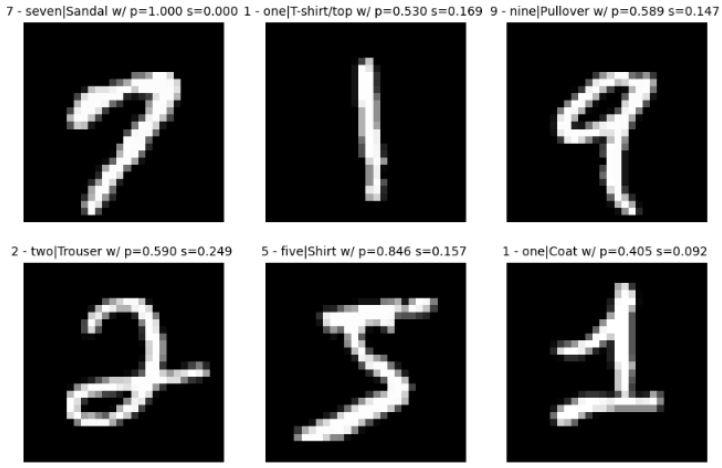

Figure 33. Visualization of ensemble predictions for out-of-distribution data, Predicted Label| True Label w/ p=mean, s = std

2.5.3.2.3 Conformal prediction

The last method explored in the case study was using conformal prediction to provide prediction sets rather than point predictions. This approach guarantees that the true label will be within the prediction set with a predefined probability, thus making predictions more robust to uncertainty. The conformal prediction method is critical in situations where making a single guess might be too risky and where offering a range of possible outcomes (with certain confidence) is more useful. The Conformal Classification library based on Regularized Adaptive Prediction Sets method is employed (Angelopoulos A. , Bates, Malik, & Jordan, 2020). Default parameters were used for conformal classification. Figure 34 provides examples of conformal sets and true labels for ID data. As observed, the prediction set does include the true class label.

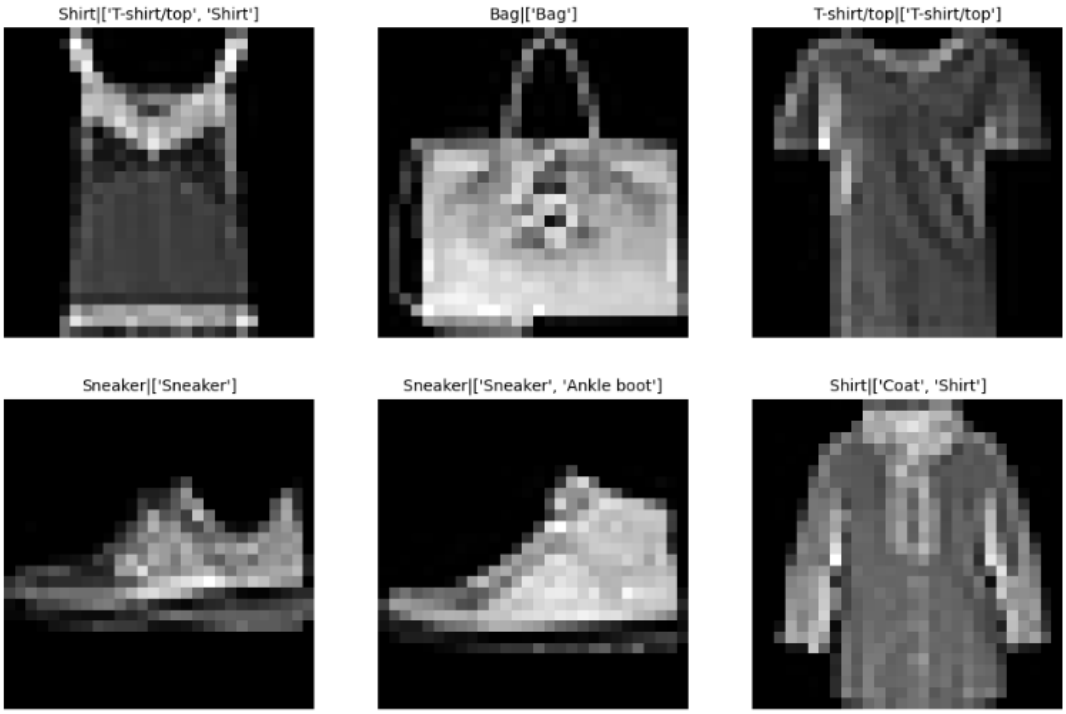

Figure 34. Visualization of conformal prediction set for in-distribution data, True label (prediction set)

Conformal prediction model was also applied to a distribution shift dataset, as shown in Figure 35. The prediction set does not include the true class label in some examples. This is because conformal prediction methods typically assume that both the training (or calibration) and test data come from the same distribution (i.e., exchangeability). This assumption breaks down under distribution shift, causing the prediction sets to lose their validity. Since the test data comes from a different distribution, the calibration may no longer represent the true uncertainty, leading to unreliable coverage (Tibshirani, Barber, Candes, & Ramdas, 2019). A similar observation was made and reasoning inferred for the OOD dataset. Methods such as weighted extension of conformal prediction as proposed by Tibshirani, Barber, Candes, & Ramdas (2019) can provide a solution to such dataset shifts.

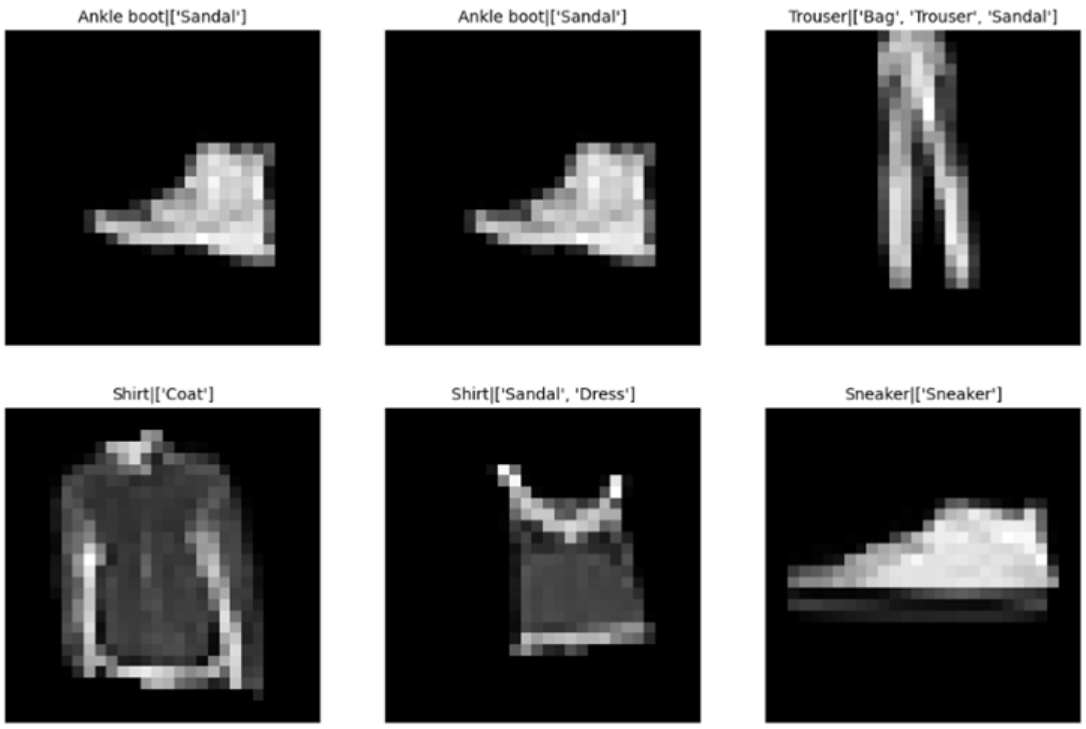

Figure 35. Visualization of conformal prediction set for distribution-shift data, True label | [prediction set]

2.5.3.3 Data-driven learned model UQ

Decisions about integration, deployment, operation, and maintenance of learned models as system components are founded on evidence of model safety, security, and performance. These decisions are made under risk by various stakeholders, including developers, service providers, regulators, and end-users.

UQ of learned-model behaviors provides a unified approach to support a variety of decisions under risks. In decision problems, the outcomes of a decision are a combination of two elements. One is the action taken by the decision maker, and the other is the relevant state of nature. The latter contributes to the outcomes, and it is uncertain and beyond the control of the decision maker. When the uncertainty of the relevant states of nature has been quantified, the decision maker is said to be facing a problem of *decisions under risk*. Otherwise, the decision maker is said to be facing a problem of *decisions under (extreme) uncertainty*. This is standard decision analysis terminology. Depending on the test and evaluation context and development state in the model cycle, we can consider different desirable or required behaviors of the model related to safety, security, and performance.

In this section, we considered two application examples based on the behavioral uncertainty quantification framework developed by Boeing (Perez & Courts, 2023).

A behavior, $\mathcal{B}$, of an entity (a system, sub-system, or component) is a restriction on certain attributes of interest (Willems J.C., 2007). For example, in an aircraft perception system that uses computer vision to detect object intrusions on the runway, a behavior may be defined in terms of the detection of a particular object type within a particular range.

Mathematically, a behavior is defined as a constraint on some manifest variables $\mathrm{w} \in \mathcal{W}$ that we can observe, namely,

$$\mathcal{B} = \{w \in \mathcal{W} : C(w)\ is\ true\},$$

where $\mathrm{C(w)}$ is a logical proposition that represents the constraint on the manifest variable.

This formal definition makes a behavior a set-theoretic concept since $\mathcal{B} \subset \mathcal{W}$. Then, if we consider the desired behavior $\mathcal{B}^*$, it is permissible to define a behavior hypothesis:

$$H\ =\ \{\,\mathcal{B}\ \subseteq \mathcal{B}^*\,\}.$$

This hypothesis is a logical proposition stating that the behavior is contained within the desired behavior. This hypothesis can be true or false (but not both), therefore we can use a verification dataset D to quantify our uncertainty about the truth or falsity of this hypothesis by computing the probability $p(H|D, I)$ conditional on the dataset and some background information I about the system or component being assessed.

The UQ problem then relates to the evaluation of the probability $p(H|D, I)$, which in this report relates to the behavior of a learned model.

In some applications, it may be necessary to consider the UQ problem under different operational contexts $\mathcal{O}_j$, where $j\ =\ 1,2,\dots,M$. These contexts may relate to

- Physical environment (e.g., weather, light, and/or electromagnetic spectrum)
- Health environment (e.g., faults and cyber-attacks)
- Relational environment (e.g., other entities in the operational space like obstacles and threats, and their relationship to the system being tested)

The UQ analysis could include the elicitation of relevant context for testing, and assign probabilities associated with these operational contexts, namely $p(\mathcal{O}_j|I)$. If this is considered, we

can then use the rules of probability to account for these scenarios and calculate a weighted average across all conditions.

$$p(H|I) = \sum_{j=1}^{M} p\left(H\middle|\mathcal{O}_j, D, I\right) p\left(\mathcal{O}_j\middle|I\right).$$

The factor $p\left(H\middle|\mathcal{O}_j, D, I\right)$ quantifies uncertainty as to how the system handles the particular operational context, whereas $p\left(\mathcal{O}_j\middle|I\right)$ quantifies uncertainty as to how likely is the system will encounter that operational context. These distributions may be used to restrict the use of the system or component only to certain operational contexts or provide input to designers to seek to improve behaviors in those operational contexts.

Note that UQ in this section is considered within the epistemic view of uncertainty. That is, it quantifies uncertainty about the behavior hypothesis conditional on a state of knowledge or information. It makes no statement about the randomness of the hypothesis. This is something that the ontological UQ cannot do (Clayton, 2021).

For details of how to make epistemic probability assignments and the computation of the sought distributions for different data types, we refer the reader to *Data-Driven Verification of Autonomous System Behaviors* (Perez & Courts, 2023), which is also provided to the FAA as an internal report.

2.5.3.3.1 Example ML classifier behavior UQ

Let us revisit the ML classifier described in Section 2.5.3.1.2. For this part of the case study, the model was retrained and the validation data set used to construct a new confusion matrix, as shown in Table 15.

Table 15. Confusion matrix

	Calibrated Model									
	T-shirt	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle Boot
T-shirt	515	0	4	12	2	1	65	0	4	0
Trouser	1	594	0	16	0	0	6	0	1	0
Pullover	15	0	490	9	57	0	24	0	0	0
Dress	14	2	6	534	17	0	28	0	0	0
Coat	0	0	43	19	490	0	42	0	0	0
Sandal	0	0	0	0	0	601	0	16	0	17
Shirt	97	1	56	21	60	0	378	0	3	0
Sneaker	0	0	0	0	0	12	0	562	0	15
Bag	2	1	1	0	2	1	5	3	551	0
Ankle Boot	0	0	0	0	0	2	0	25	0	557

For this type of problem, we can consider the following set of hypotheses:

$$H_{ij} = \{ The\ object\ O_i\ is\ identified\ as\ the\ object\ O_j \mid object\ O_i\ is\ in\ the\ image \}.$$

In this case, we consider only one operational context. Then, following the six-step procedure described by Perez & Courts (2023), we adopt a Dirichlet data model and infer the parameter distribution of this model using the Bayesian approach with a uniform prior. This prior is assigned using the Maximum Entropy Principle (MaxEnt), because we know little about this model. Then marginalized uncertainty in the parameters to obtain a posterior predictive classification uncertainty model is shown in Table 16.

Table 16. Posterior predictive probability values

Posterior Predictive Classification Probability Distrubutions										
	T-shirt	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle Boot
T-shirt	0.8418	0.0016	0.0082	0.0212	0.0049	0.0033	0.1077	0.0016	0.0082	0.0016
Trouser	0.0032	0.9475	0.0016	0.0271	0.0016	0.0016	0.0111	0.0016	0.0032	0.0016
Pullover	0.0264	0.0017	0.8116	0.0165	0.0959	0.0017	0.0413	0.0017	0.0017	0.0017
Dress	0.0245	0.0049	0.0115	0.8756	0.0295	0.0016	0.0475	0.0016	0.0016	0.0016
Coat	0.0017	0.0017	0.0728	0.0331	0.8129	0.0017	0.0712	0.0017	0.0017	0.0017
Sandal	0.0016	0.0016	0.0016	0.0016	0.0016	0.9348	0.0016	0.0264	0.0016	0.0280
Shirt	0.1565	0.0032	0.0911	0.0351	0.0974	0.0016	0.6054	0.0016	0.0064	0.0016
Sneaker	0.0017	0.0017	0.0017	0.0017	0.0017	0.0217	0.0017	0.9399	0.0017	0.0267
Bag	0.0052	0.0035	0.0035	0.0017	0.0052	0.0035	0.0104	0.0069	0.9583	0.0017
Ankle Boot	0.0017	0.0017	0.0017	0.0017	0.0017	0.0051	0.0017	0.0438	0.0017	0.9394

In this uncertainty model, each row is a predictive posterior probability mass function (pmf) conditional on the row item being the true object in the image. For example, Figure 36 shows the distribution corresponding to the first row, namely the predictive distribution that a T-shirt is classified as different objects. As we can see the classifier, as expected for such a small data set and simple ML model, can confuse a T-shirt with a shirt. This information can be used to make a decision about deployment of such a technology, or to restrict its use to particular contexts.

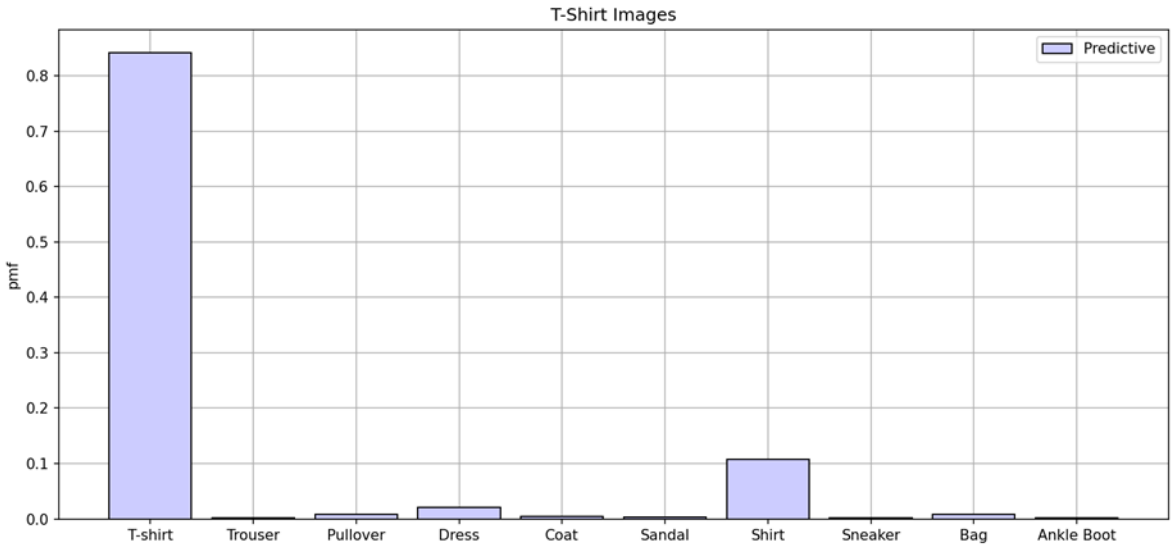

Figure 36. Posterior Predictive Distribution of a T-shirt being classified as other objects for learned model

2.5.3.3.2 Example UQ of a ML-detector behavior for runway incursion

Let us consider a learned model for a vision-based detector for extraneous object incursion on the runway during an aircraft approach to landing. We assume that such a perception system is carried on the aircraft.

The operational context may include:

- Typical objects of different sizes found at airports: utility vehicles, general aviation aircraft, jetliners, and humans.
- Weather and lighting conditions.

Figure 37 shows a sample image of a utility vehicle incursion on a runway during an aircraft approach to landing in a late afternoon with clear weather conditions.

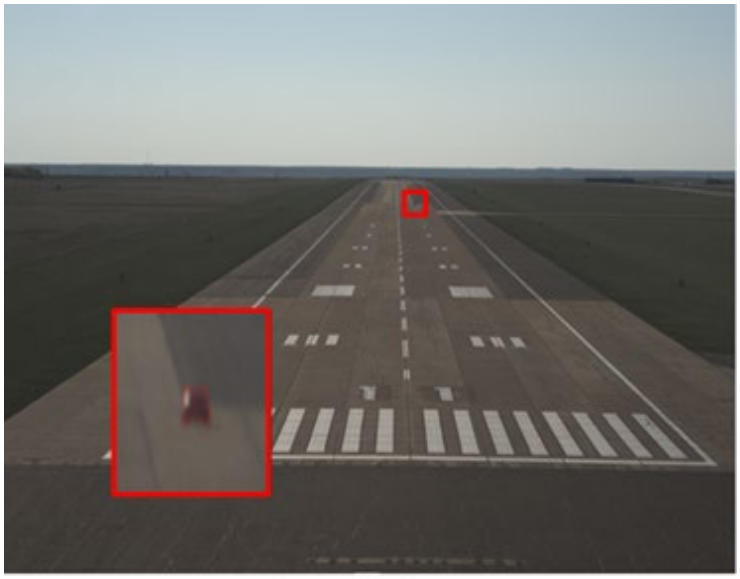

Image: Boeing

Figure 37. Sample image of a utility vehicle on a runway detected on a landing approach

A detector is a special case of a classifier with two classes, so we can consider behavior hypotheses of the form:

$$H_{1i} = \{ The\ object\ O_i\ is\ detected\ on\ the\ runway \mid object\ O_i\ is\ on\ the\ runway \}.$$

The distribution of testing this hypothesis is similar to the case of the classifier in the previous example where the validation data corresponds to images of the object over different operational contexts. Therefore, the computation of $p(H_{1i}|D, I)$ based on binary validation data is described by Perez & Courts (2023).

For this problem, however, we could also consider a behavior hypothesis that would enable the aircraft pilot to react to the object intrusion and potentially abort the landing. For this it would be of interest to consider the time-remaining to act (TRTA) after the detection and set a lower bound value t^*. This can lead to a behavior hypothesis of the form:

$$H_{2i} = \{ TRTA > t^* \mid object\ O_i\ is\ on\ the\ runway\ and\ has\ been\ detected \}.$$

Figure 38 shows some synthetic data of the TRTA over 300 detection scenarios and the predictive posterior model based on a data gamma model andMCMC. In this case, the data is continuous positive.

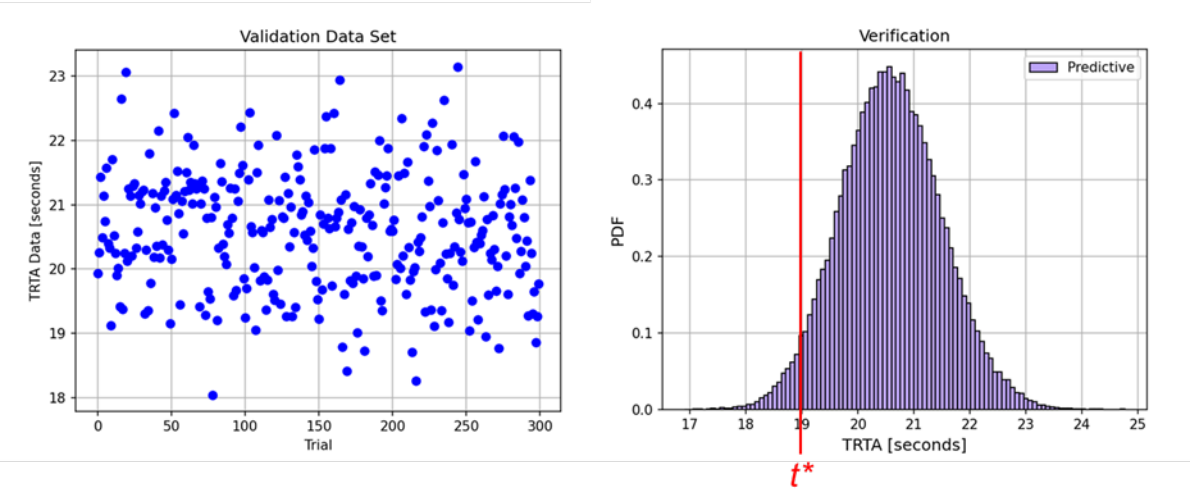

Figure 38. Synthetic data of TRTA with respect to the detection of a particular object

If we had the desired TRTA lower bound value $t*$, we would be able to compute $p(H_{2i}|D,I)$. However, we have left $t*$ unspecified. There are some examples where the behavour may be partially specified a priori, as in this example. Then, the predictive model may be used to complete the specification.

This section provides an overview of an approach to UQ post training of learned model behaviors using epistemic uncertainty. The results obtained can support various stakeholders' decisions under risk at the time of integration and prior to the deployment of learned models.

2.5.3.3.3 Case study conclusion

The key takeaway from this case study is that deep ensemble and conformal prediction approaches can improve the model's ability to handle uncertainty, particularly when working with OOD data. Techniques like vector scaling and conformal prediction can also be applied to improve model reliability in real-world tasks where uncertainty is prevalent. However, implementing these techniques will not work without modification and need further investigation. An iterative and investigative approach is required.

The baseline model performance could have been improved by additional training epochs, augmenting the dataset, or changing the model architecture (Taiga, Buchanan, Pleiss, Zemel, & Cunningham, 2022) to have similar performance as ensembles. Further, the hyperparameters for the training baseline models, ensembles, and conformal prediction could also be tuned to yield better performance, but the goal of this case study was merely to introduce the workflow of UQ

using a simple model. When such a workflow is used for aviation applications, many safety considerations must be accounted for.

A brief overview of learned model behavior UQ is provided, and two examples are shown. As discussed in (Perez & Courts, 2023):

- Behavior UQ offers a unified framework for generating evidence about the safety, security, and performance of learned models.
- Behavior definitions derived from the customer's mission engineering threads.
- Behavior definitions consistent with standard practices in systems engineering (INCOSE) (Walden, et al., 2023).
- Behavior mathematical models enable formulation of appropriate behavior hypotheses.
- Hypotheses are considered within an epistemic UQ approach (probabilities reflect incomplete knowledge).
- Epistemic UQ analysis provides the best UQ based on the available data and knowledge.

2.5.4 Uncertainty management challenges and recommended practices

UQ in ML faces several challenges, notably in distinguishing between aleatoric and epistemic uncertainty. Aleatoric uncertainty, inherent to data, cannot be reduced even with more data or model improvements, while epistemic uncertainty, stemming from model limitations, can potentially be minimized. In practice, however, due to the layering of complexity of different sources of uncertainty, as shown in Figure 39, distinguishing each is very difficult. In current practice it is assumed that all uncertainties are epistemic and therefore require more data collection.

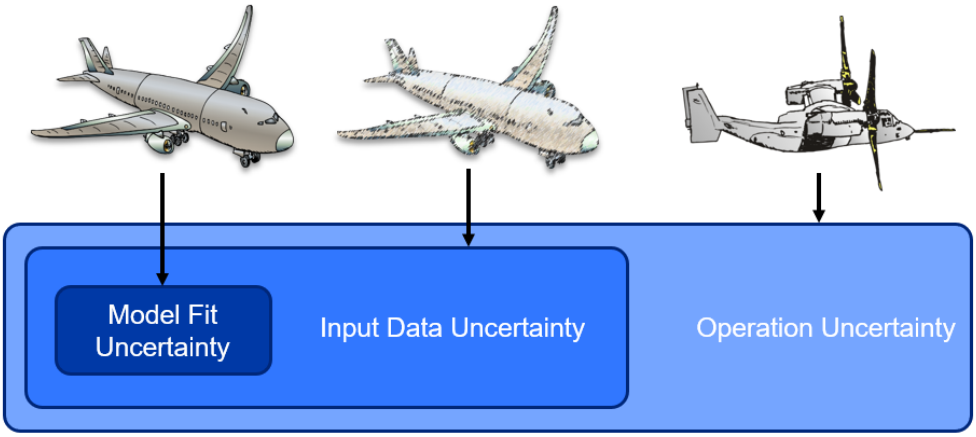

Figure 39. Layered complexity of uncertainty

The absence of ground truth complicates efforts to accurately estimate or validate these uncertainties. Additionally, the integration of ML models into feedback loops poses further complications. Explainability is also an issue with how models derive their uncertainty and is crucial for reliability and usability. Lastly, the computational complexity associated with scaling for some calibration methods grows significantly, with linear increases for vector scaling and quadratic increases for matrix scaling as the number of classes expands, creating a substantial burden on processing resources in complex models.

Recommended practices for UQ include focusing on quantifying uncertainty in individual system functions to gain a detailed understanding of specific areas where uncertainty arises. If available, comparing model outputs to theoretical performance benchmarks, for example in systems like cruise altitude recommendation for fuel consumption or the Airborne Collision Avoidance System (ACAS), helps validate a model's accuracy. Additionally, conducting thorough data and architecture reviews to identify and address outliers is crucial, as these can significantly impact model performance. Finally, reconstructing the distribution of model outputs allows for a more robust representation of uncertainty, providing deeper insight into the model's predictive behavior.

2.6 Safety assessment of ML-enabled systems

To ensure safety, reliability, and performance of aircraft, V&V activities are conducted in parallel to system development. Validation activities include Aircraft Functional Hazard Analysis (AFHA), Preliminary Aircraft Safety Analysis (PASA), System Functional Hazard Analysis (SFHA) and Preliminary System Safety Analysis (SAE, 2023; ARP-4754 B, Guidelines

for Development of Civil Aircraft and Systems, 2023). These activities ensure aircraft safety, show compliance with regulations, and assist in development and compliance with internal standards.

A typical program at Boeing for system validation starts with planning and drafting a Validation Plan, including all tiers of airplane and system-level requirements. The Validation Plan is contained in the Plan for Development Assurance Aspects of Certification (PDAAC), which is submitted to the FAA for acceptance. The Validation Plan has several commonalities with other planning documents such as the Requirements Management Plan, Verification Plan, and Configuration Management Plan. Therefore, the Validation Plan is typically not a standalone document but rather integrated with one or more planning documents.

The validation planning phase establishes the framework that is later used for completing and documenting validation activities. In this phase, the validation plan is developed to identify the processes and standards that will be used for requirements validation. Here it is decided on the validation methods, personnel involved, and level of rigor.

After the planning phase, the validation activities are performed. At the time of authoring system requirements during the development phase of the program, the design team may identify peer and subject matter expert review along with test or analysis as validation activities that are necessary to support the requirement validation. The intent of these tests and analyses is to support requirement validation, so the artifacts generated from these activities must provide sufficient evidence to justify requirement definitions and rationales.

This section of the report focuses on a study conducted to assess SFHA performed for an ML-enabled system. SFHA is used for identifying and assessing the failure conditions associated with system functions. It is the primary methodology for setting hazard classifications associated with a system, and the results are the primary consideration in the development of a compliant system architecture.

2.6.1 Review of safety assessment methods for ML-enabled systems

There is a growing imperative among ML practitioners within the aviation community to proactively incorporate safety considerations during the early stages of designing safety-critical systems that involve ML components. Given the relatively nascent state of safety assurance pertaining to ML constituents, a comprehensive review of the literature reveals a scarcity of publications that specifically address safety assessment during the validation phase for aerospace applications.

Data constitutes as an important input to the ML pipeline and plays a vital role in providing safety assurance of the system that uses ML components. A data governance framework is proposed by Carter, Chan, Vinegar, & Rupert (2023) to provide data assurance. This ensures test data set correctness, completeness, and representativeness of operational design domain. Data assurance, along with learning assurance, compensate for the loss of traditional software development objectives such as loss of low-level requirements, traceability, etc. An important element of this data assurance framework is data hazard analysis. Failure Mode Effect Analysis (FMEA) is proposed as one type of data hazard analysis method. ML data FMEA could be used to assess the failure effects of the data on the ML model. Such kinds of data hazard assessment would be conducted prior to data collection and source identification to inform and ensure collection of safety critical features in data.

During the review process, a large number of safety analysis publications emerged from the automotive domain. A safety analysis technique using Hazard and Operability Analysis is proposed by Qi, Conmy, Huang, Zhao, & Huang (2022) for an ML-enabled system. The approach breaks down the ML-enabled system hierarchically into three levels: the System Level, ML-Lifecycle, and Inner-ML Level that performs Hazard and Operability (HAZOP) analysis at each of these levels. Using the HAZOP method hazards, causes and mitigations are identified.

Abrecht, Hirsch, Raafatnia, & Woehrle (2023) presents a method for addressing safety concerns related to DNNs in automated driving systems. The functional insufficiency of DNNs is attributed to a combination of triggering condition and safety concerns. The safety concerns are categorized into four categories: open world context, data and dataset preparation, DNN characteristics, and analysis and evaluation. By refining and categorizing the safety concerns with input from domain and safety experts, the authors develop a structured approach to systematically identify and analyze safety risks.

Existing methods utilized for safety analysis exhibit certain shortcomings. For instance, the use of FMEA for data and its impact on the performance of ML models presents significant challenges (Carter, Chan, Vinegar, & Rupert, 2023). The provided example focuses on a limited subset of features, indicating a potential lack of comprehensive analysis. Despite extensive research on the effect of data on ML models, there are still areas, such as data quality, quantity, diversity, and distribution, that require further exploration and remain insufficiently understood.

Another notable challenge arises from the proposal of using Bayes Net as a quantitative method for safety analysis of ML models (Qi, Conmy, Huang, Zhao, & Huang, 2022). Defining the

probability values for Bayes Net poses difficulties, particularly when dealing with wrong class predictions by the ML model.

These challenges underscore the complexity of performing safety analysis and emphasize the pressing need for further research.

2.6.2 Fault modeling design tool (fmdtools)

The fmdtools toolkit is a simulation toolkit that allows the modeling of dynamic effects of hazardous events, developed by the Robust Software Engineering group at NASA Ames (NASA, 2024). This toolkit uses a function-flow based system representation to represent and propagate failures, enabling it to be used in early design hazard assessments like functional hazard assessment. The functions in fmdtools represent parts of systems that have behaviors (e.g., agents such as operators, components, and technical functions such as storing energy). The flows represent energy, materials, signals, information, or any other system artefacts that are shared among functions. This function-flow-based representation allows the representation of components, software, and human elements of systems in the hazard simulation. The toolkit allows the parametrization of models and contains prebuilt sampling approaches, allowing the study of system behavior over a large number of potential hazards that can be internal or external to the system or operational in nature. Past applications of fmdtools have studied hazards in individual systems (e.g., rovers, drones, etc.) and systems of systems (e.g., taxiway operations and ariel wildfire response). However, it has never been applied to study ML components.

2.6.2.1 Purpose

The primary goal of this study is to explore the utility of fmdtools for performing functional hazard assessment of perception-based ML-enabled system, AutoTaxi Perception System. In the past, fmdtools has been used for hazard assessment of complex systems and systems of systems. We aim to explore if the existing tools in fmdtools can sufficiently perform a hazard assessment of perception-based systems and identify any gaps in the toolkit for future development.

2.6.2.1.1 Metrics

To evaluate the risks posed by perception faults in different scenarios, a variety of novel metrics are proposed. They are object criticality, object risk category and cost of lost information (COLI).

Object criticality (perceived and real) was defined using the equation:

$$C_O = 1 + \frac{9.0}{e^{\frac{t}{10*w}}}$$

Where t is the time to the object (based on its relative velocity) and w is a weighting factor based on the type of object (1 for aircraft, 0.5 for ground vehicles, 0.2 for ground points, and 0.3 for all other objects). Note that the exponential scaling means that object criticality ranges from 1 to 10.

Object risk category used the ontology shown in Figure 40. In this ontology, any object with an $5\ s$ action window or smaller is categorized as a "critical perception." If an object is outside the $120\ s$ action window, it is categorized as "supportive perception." All other objects are categorized as "functional perception." These time windows were assumed based on average $20\ knots$ speed of aircraft during taxing operation, resulting in a $\sim 50\ m$ and $\sim 1200\ m$ distance travled in $5\ s$ and $120\ s$ action window, respectively.

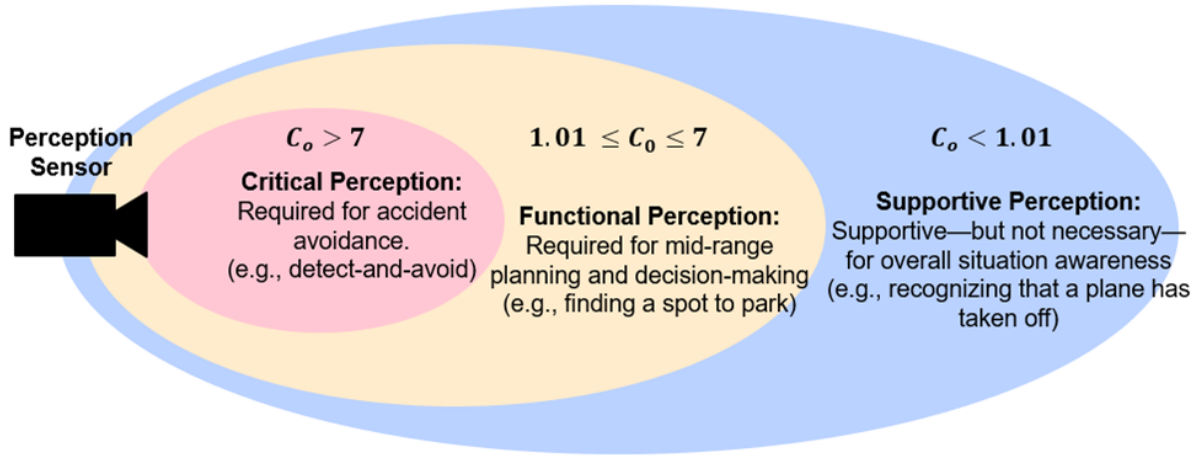

Figure 40. Object categorization based on object criticality

To categorize the overall severity of a fault scenario, two main types of misperceptions were defined for the model as listed below:

- *Object perceived incorrectly*: In this scenario, an object in the environment is perceived, but with incorrect data.
- *Nonexistent object perceived*: In this scenario, an object that is not truly in the environment (e.g., a ground point or hallucinated point) is perceived.

These two misperceptions have two different outcomes associated with them. If an object is perceived incorrectly, it may not be avoided; whereas if a nonexistent object is perceived, it may lead to false avoidance. Since hitting an object is usually worse than avoiding a nonexistent object, these misperceptions are classified somewhat differently to develop the overall severity classification. The severity classification levels are shown in Table 2-16 and are analogues to

qualitative classification provided in AC 25.1309 draft ARSENAL (Aviation Rulemaking Advisory Committee (ARAC), 2002).

Table 17. Severity classification defined for fmdtool analysis

Classification of Severity Level	
Severity Level	**Incorrect Object Class or Nonexistent Object Class**
Major	Critical
Minor	Functional
Negligible	Supporting

COLI, an information-theoretic metric, was developed to categorize the impact of perception uncertainty on object priority assignment for downstream systems. COLI is calculated using the following equation:

$$COLI = \max_{o} C_{O,a} - C_{O,a}\left(\arg\max_{o} C_{O,p}\right)$$

Where $C_{O,a}$ is the actual criticality of an object o and $C_{O,p}$ is the perceived risk of an object o. In essence, this metric compares the *actual risk* of the object perceived to be highest priority by the system with the risk of the object that *should be* perceived as highest priority. This metric and supporting metrics are shown in the results for the analysis of the model. Metrics like COLI can help study the varying effects of a hazard in different scenarios. While we only account for the worst-case outcomes in the functional hazard assessment, as more detailed safety assessments are performed and system requirements defined, metrics like COLI can inform if the worst case outcome is an outlier or if it is a more common phenomenon when analyzed over different scenarios, informing mitigation strategies.

2.6.2.2 Tools conclusions

As discussed in the previous sections, fmdtools was used to perform the analysis of the ML-based subsystem, starting from a component architecture. While this bottom-up approach to the analysis can inform about the hazard severity relating to specific subsystems, it is contrary to the top-down approach the fmdtools is intended to be used in. With the top-down approach, fmdtools can be used to analyze hazards starting from when high-level functions are identified until component level designs are developed. Following the top-down approach aligns more with traditional safety assessment approaches and maintains traceability throughout the analysis, informing the safety assessments.

One trade-off to consider when modeling using fmdtools is model granularity versus time spent developing the model. Setting up detailed models in fmdtools can be time consuming. In this case study, we spent more time making the models more detailed instead of focusing on creating simple models that are sufficiently detailed to aid the hazard assessment, because one of the goals of this work was to showcase the some of the capabilities of fmdtools. Some of the capabilities showcased in this study include:

- Ability to perform a model-based hazard assessment as opposed to the traditional document-based approach, enabling more consistency and traceability with other model-based design artefacts.
- Ability to define quantitative metrics that are tailored to the system and assessments performed (e.g., defining the metric cost of lost information to represent the effects of a perception failure).
- Ability to visualize failure propagation, failure over time graphs, scenarios, and combinations of these visuals to gain a more comprehensive understanding of the effects of failures.

This study can be expanded further to perform statistical, sensitivity, and behavior assessments. The fmdtools toolkit allows the assessment of a large sample of scenarios over different faults (including joint faults) and system parameters. For example, this study can be expanded by studying fault propagation over a variety of different maps (e.g., using training data), studying the sensitivity of the perception components to varying taxiway configurations, or answering statistical questions such as “Is the perception system more vulnerable to failures around hangars or dedicated taxiways?” Moreover, a function may be realized through a variety of behaviors. In fmdtools, these behaviors can be varied, and the overall outcome on safety can be measured. All these studies can inform system requirements and specifications in addition to informing hazard assessments.

2.6.3 Conclusion

The safety analysis of ML components poses significant challenges that must be carefully addressed to ensure the safety the system. One of the primary challenges in safety analysis is the comprehensive capture of failure modes in hazard analysis due to the emergent behavior of ML enabled systems. Deployment of such systems in safety-critical applications often operate in dynamic and uncertain environments, making it even more challenging to pinpoint the exact causes and consequences of failures. However, a functional description and leveraging

understanding of the ML-enabled system can greatly aid in the identification of potential failure modes. By understanding the system's intended behavior and its underlying algorithms, it assists in identifying potential failure scenarios and their associated risks.

Addressing the challenges in safety analysis of ML components requires a multidisciplinary approach. Collaboration between domain experts, data scientists, and safety engineers is crucial to ensure a comprehensive understanding of the system's behavior and potential failure modes. Additionally, the development of standardized methodologies and guidelines specific to ML safety analysis can help streamline the process and ensure consistency across different applications and industries.

In conclusion, while the safety analysis of ML components presents challenges, it is essential to overcome them to ensure the safe and responsible deployment of these systems. By continuously improving our understanding of ML failure modes, leveraging functional descriptions, and utilizing specialized tools, we can enhance the safety and reliability of ML-enabled systems in safety-critical applications.

3 Risk mitigation approaches in AI/ML systems

AI/ML components offer tremendous capabilities in various domains, such as computer vision, fault-tolerant control, decision making, and many more. However, when these components are deployed in safety-critical systems, there exist a lot of challenges to certify them (SAE, 2021). Along with the certification challenges, there is also a significant safety risk due to various types of failure conditions of ML components (Kumar, O'Brien, Albert, Viljöen, & Snover, 2019). These failure conditions and modes can be broadly classified as unintended or intentionally motivate failure modes, and they are initially classified as follows, with several examples shown in Figure 41:

Unintended Failures

- Generalization and reproducibility (Yang, Zhou, Li, & Liu, 2022) (Figure 41a)
- Uncertainty and distributional shift of data
- Risk related to hardware supporting execution of AI (Figure 41c), e.g., sensor, hardware, GPU etc. (Kumar, O'Brien, Albert, Viljöen, & Snover, 2019)
- Incomplete testing

Intentionally Motivated Failures

- Perturbation attack
- Model inversion
- Adversarial examples in physical domain (Eykholt, et al., 2018) (Figure 41b)
- Backdoor machine learning

Figure 41. (a) OOD data; (b) Physical adversarial perturbations; (c) Class prediction confidence reduction due to image distortion

3.1 Run-time assurance and contingency management-based approach

Safety risk mitigation mechanisms are critical for AI/ML-enabled autonomous systems due to limitations of design-time V&V techniques, like scalability and completeness, or unexpected uncertainties in the system or environment. Current well-adopted risk mitigation approaches for complex systems include RTA, CM, Fault detection, Isolation, and Recovery (FDIR), health management, controller synthesis, fault tolerance, redundancy, and safety pilots.

AI risk mitigation techniques have been proposed which either involve human-in-the-loop (Torens C. , et al., Machine Learning Verification and Safety for Unmanned Aircraft - A Literature Study, 2022), operation design domain (Torens C. , et al., Ensuring Safety of Machine Learning Components Using Operational Design Domain, 2023), black-box simplex architecture for run-time assurance (Mehmood, Sheikhi, Bak, Smolka, & Stoller, 2022), avoiding end-to-end deep learning method (Perrotin, Roy, Herbulot, Devy, & Bousquet, 2022). The proposed risk mitigation approach comprises integration of two discrete elements: AI RTA and CM. The approach has two discrete and separate components to minimize architectural complexity and difficulty in defining requirements, compared to a single unified component. The proposed risk mitigation method and its implementation are expected to provide assurance and manage ML-related failure modes.

3.1.1 Review

Operational safety of avionics software traditionally was ensured by a rigorous V&V process and testing procedures. However, with recent integration of MLcomponents in avionics, new challenges emerge to ensure operational safety. Traditional testing procedures do not apply to

address these new challenges. This section reviews methods and architectures of RTA and CM, which may individually or in combination be an alternative to assure operational safety.

3.1.1.1 Run-time assurances

A method to provide assurance of a complex function during run time is proposed in the ASTM F3269-21 standard (ASTM, 2021). This practice provides an architectural framework to bound the behavior of the complex system or ML component. The RTA functional architectures are shown in Figure 42 (Dunlap, Mote, Delsing, & Hobbs, 2022) and Figure 43 (ASTM, 2021). The building blocks of this RTA are input manager, safety monitor, recovery functions, and RTA switch. Various case studies implementing the ASTM RTA architecture are also presented.

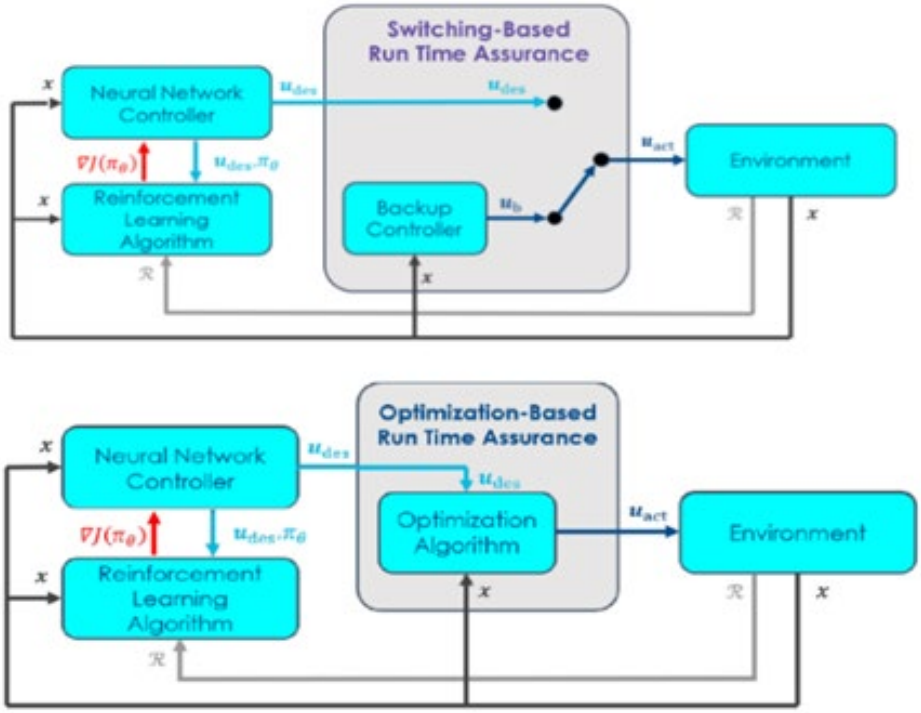

Figure 42. RTA architectures for reinforcement learning

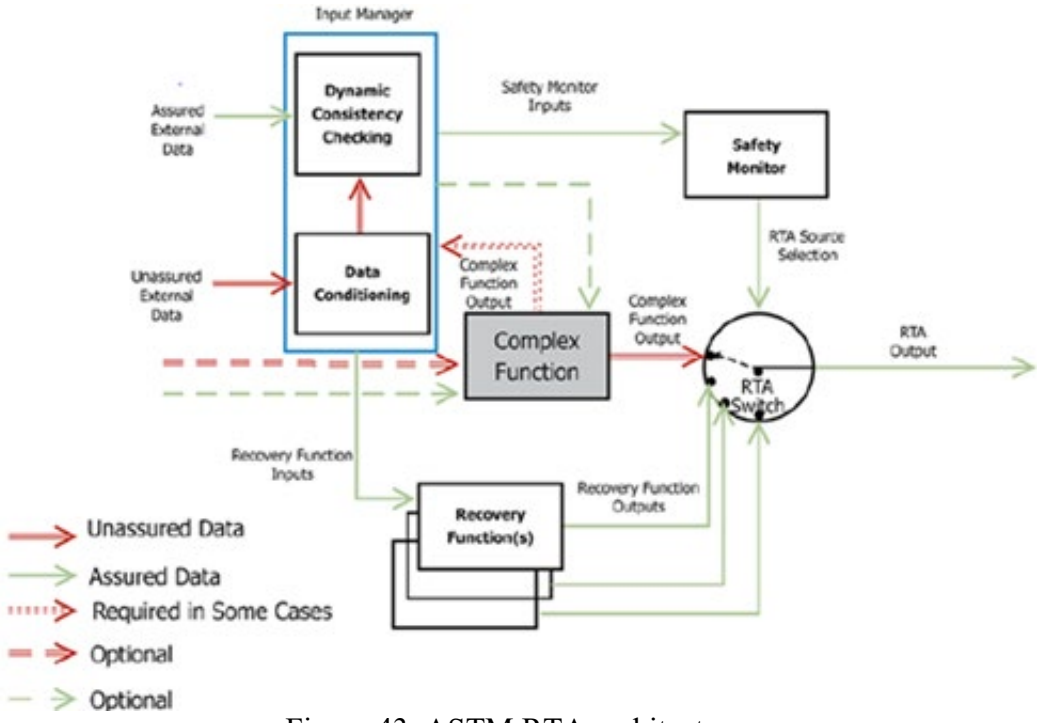

Figure 43. ASTM RTA architecture

Approaches to bound the behavior of a reinforcement learning agent to learn safely are proposed by Dunlap, Mote, Delsing, & Hobbs (2022) and shown in Figure 42. The first approach is a switching-based RTA filter, which evaluates if the input to the agent provided by the neural network controller is safe or not, and if unsafe it switches to a back controller. The second approach utilizes barrier function constrained online optimization to assure safety of the system while learning. The proposed methods are applied to an aircraft docking application in simulation. As seen from the results, there is a relative increase in training time compared to traditional RL without integrated RTA. But the utilization of RTA provides safety guarantees during learning, which cannot be guaranteed otherwise. Also, among the two RTA methods, an optimization-based method provides a more optimal behavior but traded off with a higher computation time compared with a switching-based method.

Cofer, et al. (2020) have demonstrated utilization of RTA to safely operate a learning-enabled component (LEC). The functional architecture of this demonstration is shown in Figure 44. The LEC is implemented with a DNN trained to estimate cross-track error (CTE) of the aircraft (position left or right to the runway centerline) based on images from a forward-looking camera on the aircraft. Multiple safety monitors are used to ensure safe behavior of LECs, including GPS signals to compute aircraft position relative to virtual runway, runway centerline detection using

CV, high integrity inertial reference system and a variational autoencoder used to determining if LEC is operating in a region of competence relative to training data. An important conclusion from the demonstration is that it was not always possible to monitor the variables or conditions needed to detect safety violations. And in some cases, it is not obvious how to create a safe backup function that is less complex.

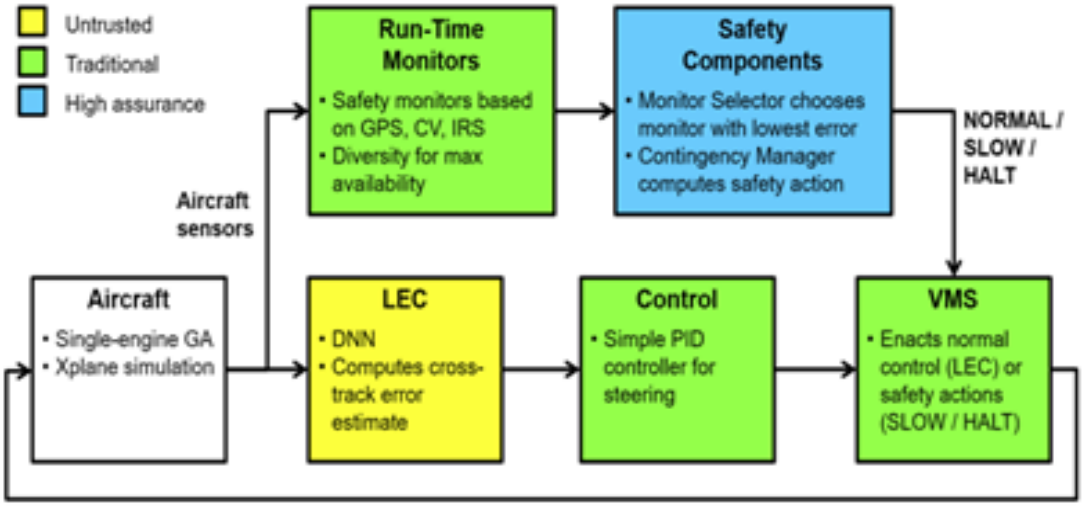

Figure 44. TaxiNet demonstration system with RTA architecture

An RTA architecture which expands on the ASTM RTA architecture is proposed by Skoog, Hook, & Ryan (2020) and shown in Figure 45. This architecture is called Expandable Variable Autonomy Run-Time Assurance Network (EVAA RTA). The key idea is that system flexibility and robustness in the RTA design can be achieved through an RTA network. The novelty elements of the EVAA architecture include multiple monitors to cover all intended safety functionalities, a switch identifying extremely critical safety scenarios, and a second RTA switch for re-planning due to contingencies.

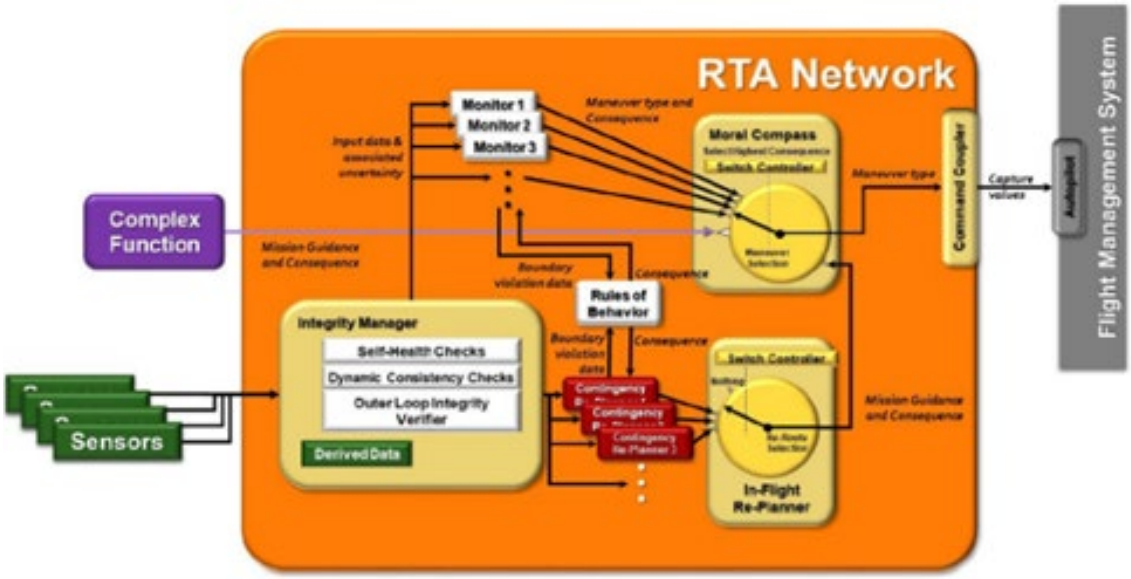

Figure 45. EVAA RTA network architecture

Various types generic RTA architectures are introduced in the tutorial on developing RTA systems (Hobbs, Mote, Abate, Coogan, & Feron, 2022). These architectures include Simplex architecture, Active Set Invariance Filters (ASIFs), and their respective implicit/explicit variations. Simplex RTA based on decision logic which switches between primary controller and backup controller, and is system agnostic. Active Set Invariance Filter (ASIF) is associated with optimization-based algorithms and results in a smoother and more gradual intervention than approaches that rely on switching to backup controller.

3.1.1.2 Contingency management

In (Prashin Sharma, 2022), author has proposed a contingency manager which recommends safety-preserving actions for the UAS utilizing abstracted information related to flight status, remaining flight plan, and component health indicator. The contingency manager is formulated as an MDP computed offline. Actions from the contingency manager include executing emergency landing as soon as possible, or when possible, or pop a parachute. It is integrated into the UAS Guidance, Navigation, and Control, as shown in Figure 46 (Prashin Sharma, 2022). Effectiveness of the contingency manager is evaluated for UAS package delivery missions and has shown to significantly reduce the failure rate of the mission in scenarios where there is motor failure or degraded battery conditions. In follow-on research, authors have proposed a Partially Observable Markov Decision Process (POMDP) model (Sharma, et al., 2023) of the contingency manager. The POMDP model enables simulating sensor noise in the system and evaluates the performance of the contingency policies.

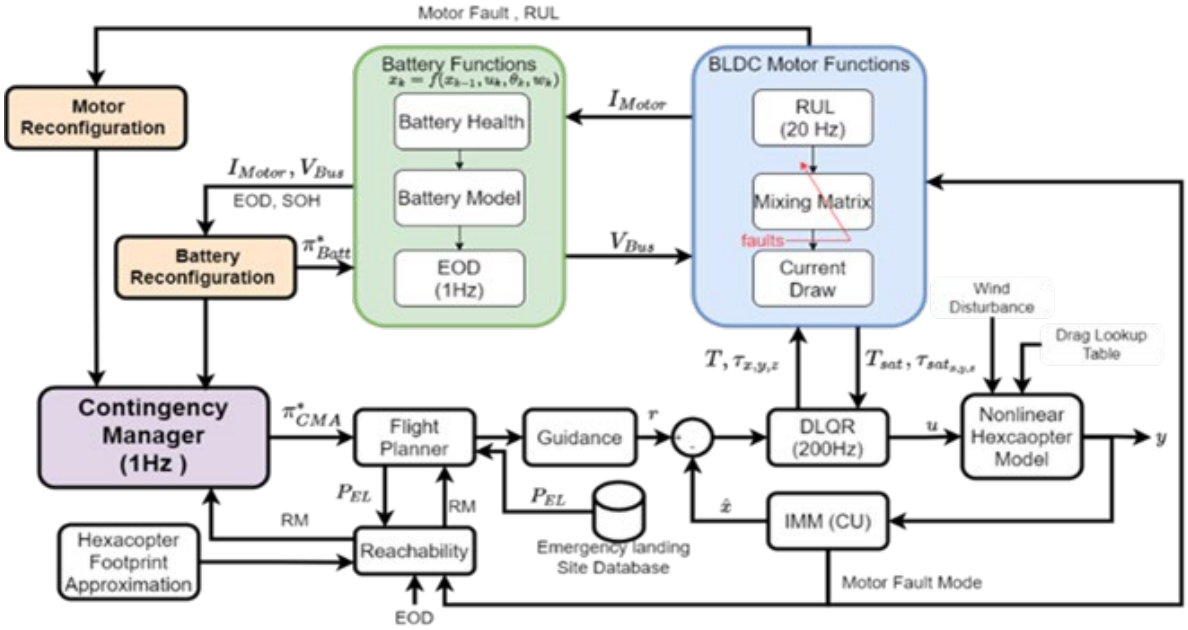

Figure 46. UAS CM architecture

An intelligent approach to contingency management is proposed by Gregory, et al. (2021) to provide a more general coverage of vehicles and operating environment contingencies in a safe and cost-effective manner, as shown in Figure 47. The presented approach of an intelligent contingency manager considers some emerging techniques in ML and explores their integration into safety-critical systems and potential levels of assurance. A vehicle capability assessment module assesses vehicle's current state and predicts the state of the vehicle into the future (Campbell, Acheson, & Gregory, 2021). Mission execution controls all aspects of vehicle flight. There is a feedback loop back from mission execution to vehicle capability assessment.

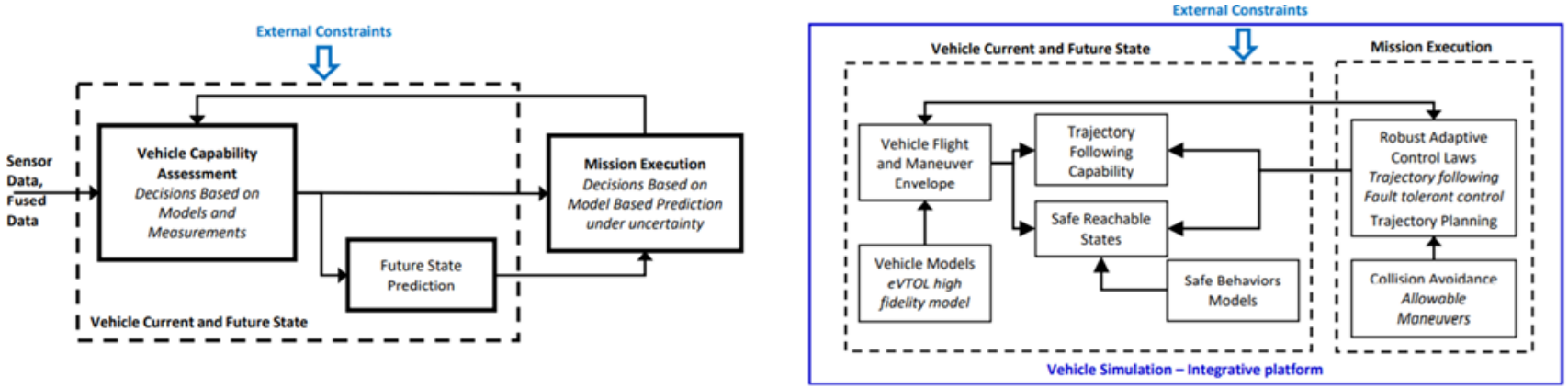

Figure 47. Intelligent contingency manager

The contingency planning framework proposed by Zhou & Kwan (2018) tackles lost link fault. The proposed contingency planner has four main functions:

- Risk Management Plan (RMP), Rescue and Recovery Plan (RRP), Preparedness and Prevention Plan (PPP) and Incident Response Plan (IRP), as shown in Figure 48. The RMP analyzes the risk due to lost link fault. The PPP module generates lost link plan accounting for UAV mishaps. IRP tracks the flight status and predicts trajectory of UAV under distress. Lastly, RRP module renews the link when the UAV arrives at the rendezvous point.

Procedure for various modules of the lost link contingency planner are generated offline and stored in a database and used online. The database also includes FAA regulations, UAV performance models, airspace structure, ground radio towers, airport locations, flight termination points, rally/reconnection waypoints, and hovering waypoints near airports. These databases are used during online plan selection.

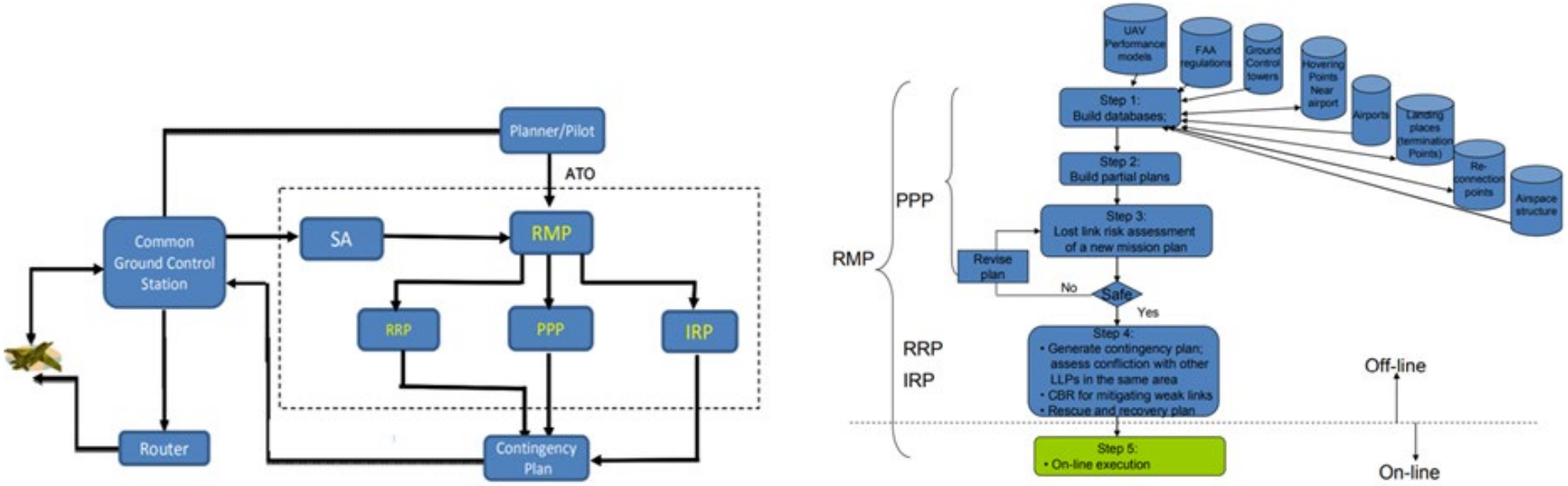

Figure 48. Contingency planner architecture and procedure developing lost link contingency plan

A model-based run-time architecture is presented by Schumann, et al. (2019), used for on-board diagnosis, system health management, and CM. A Decision Maker module is included in the architecture tasked with determining the impact of the failure and its disambiguation, along with determining a suitable contingency plan needed to recover from the failure and fly the aircraft safely. With ambiguities resolved, the contingency planning is started. The DM employs a recursive schema-based planning algorithm, which uses backtracking search, constraint logic, and logic-based programming.

3.1.2 Generic AI RTA architecture

This section introduces an architecture of RTA for AI/ML functions in a larger system. The purpose of AI RTA is to ensure that the behavior of AI/ML function in the system is monitored, overall system behavior is bounded, and assurance guarantees are provided during run time. The following sections describe the functional architecture and requirements of RTA.

3.1.2.1 Functional architecture

The functional architecture of the AI RTA is shown in Figure 49. The input to the AI RTA is the sensor data, and output is either from AI/ML function or the verified function.

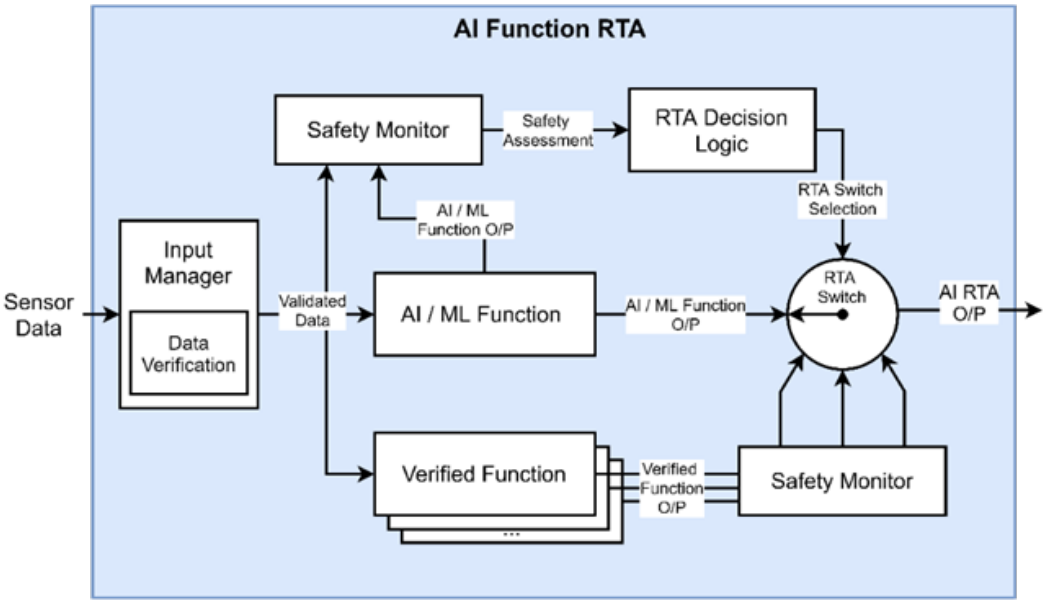

Figure 49. Functional architecture for AI RTA

Modules of the AI RTA are defined as follows:

- *AI/ML Function*: A function to which traditional verification methods are difficult to apply and which cannot be assured during design time.
- *Verified Function*: A function which can be verified during design time and provide an output that will maintain safety of the overall system.
- *Input Manager:* A module that ensures downstream RTA modules receive assured data.
- *Safety Monitor*: A module that evaluates the AI/ML function behavior or the verified function by directly monitoring the output or input or both.
- *RTA Decision Logic*: A module that informs the switch to select between one of the AI/ML or verified functions.
- *RTA Switch*: A means to switch between different AI/ML and verified functions.

The Input Manager module ensures the downstream RTA modules receive assured data by performing verification on sensor data. This validated data is sent to the AI/ML function, verified function and the safety monitor. There exist multiple safety monitors in the AI RTA architecture. The first safety monitor inspects the input or optionally the AI/ML function for unsafe input or undesirable behavior. An example of unsafe input is OOD data for a ML vision system, and undesirable behavior could be unwarranted firing of neurons in a neural network. The second safety monitor checks the output of the verified function. The objective of both safety monitors is to ensure system safety. The AI/ML function performs inference on the validated data and serves it to the RTA switch. Similarly, the verified function generates an output and passes it to the safety monitor. RTA decision logic utilizes the safety assessment information from all the monitors and directs the switch to select a risk-free input to be the output of AI RTA.

3.1.2.2 AI RTA architecture

The ASTM RTA standard has defined requirements for RTA (ASTM, 2021). For the application of the proposed AI RTA, available RTA requirements are synthesized to be defined at the architecture and module levels as follows:

Architecture requirements

- Adverse transients do not occur while switching between different sources of RTA output.
- Define bounds to ensure that the AI/ML function remains in a Safe State.

- Define a threshold beyond which the Safety Monitor activates a verified function that ensures the RTA output within predefined bounds.
- The designer shall ensure RTA component failures are detected and handled.

Input manager requirements

- The input manager shall perform data verification on all sensor data.
- The input manager shall compute and output data quality attributes for all input manager outputs, such as update rate, accuracy, precision, and latency.

Safety Monitor and RTA Decision Logic requirements

- The Safety Monitor shall continuously evaluate the AI/ML function behavior by monitoring Input/Output, or the RTA Output, or system behavior or any combination.
- The Safety Monitor shall detect behaviors that indicate that the AI/ML function will leave the Safe State.
- The Safety Monitor shall detect output from the verified function when the system leaves the Safe State.
- When the Safety Monitor thresholds are exceeded, the RTA Decision Logic shall select and activate a suitable Verified Function to ensure that the predefined bounds are not violated.

RTA switch requirements

- It shall only accept RTA Source Selection from the RTA Decision Logic.
- It shall ensure that the only source of the RTA Output is the function (Complex Function or Verified Function) selected by the Safety Monitor.
- It shall ensure that there is always exactly one source of RTA Output at all times.

Verified Function requirements

- It shall ensure Verified Function output is available at time of selection.
- The Verified Function should inform the Safety Monitor when the Verified Function is unable to provide and output.

3.1.3 Generic CM architecture

CM is the ability to mitigate risks in critical states or events while preserving safe operations of the aircraft. CM could be triggered by either states or events. *States* are conditions or modes internal to the system which can or cannot be controlled, for example equipment failure (propulsion system failure, navigation sensors failure, GPS failure, flight controller failure, battery failure, control surface failure), lack of fuel, loss of communication with the Ground Control System (GCS), etc. *Events* are external factors that affect the system and cannot be controlled, for example weather conditions, obstacles, terrain, adversarial aerial vehicle, bird strike, etc. The overarching requirement of CM is to recognize each contingency situation and respond appropriately. This section introduces a generic CM architecture for AI/ML functions.

3.1.3.1 Functional architecture

The functional architecture of the contingency manager is shown in Figure 50. The input to the contingency manager is validation data of single/multiple sensors and output from an AI function.

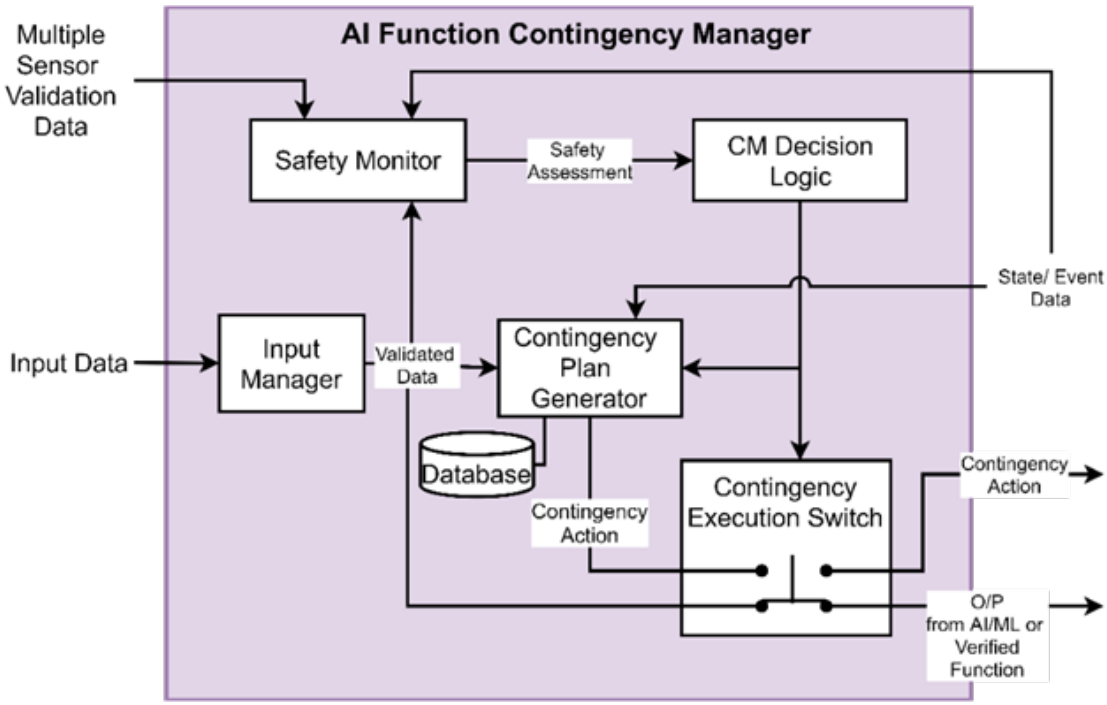

Figure 50. Functional Architecture for AI/ML Contingency Manager

Modules of the contingency manager architecture are defined as follows:

- *Input Manager:* A module that assures downstream CM functions receive assured data.

- *Safety Monitor:* A module that assesses safety of the system based on state/event information, validated data from the input manager, and conditions of the sensors used by the AI function.
- *Contingency Decision Logic (CDL)*: A module that triggers planning and execution of the contingency plan.
- *Contingency Plan Generator (CPG)*: A module that generates a contingency plan.
- *Contingency Execution Switch (CES)*: switches between nominal system operations and contingency plan execution.

The output of the AI function acts as an input for the AI contingency manager. This input data is validated by the input manager, and the validated data are sent to the safety monitor, CPG, and CES. The safety monitor assesses the safety of the system based on the information it receives about the state of the system, events, validated data, and sensor condition. This safety assessment is processed by CDL to identify if a contingency action is needed or to carry out nominal operations. If the CDL identifies a contingency scenario, it informs the CPG to generate a contingency plan. The CPG utilizes the state/event information and available database to generate a plan based on contingency identified by the CDL. In a contingency scenario, the CDL also informs the CES to switch to the contingency plan.

3.1.3.2 Contingency management requirements

Requirements of CM have been proposed by Teomitzi & Schmidt (2021), Usach, Torens, Adolf, & Vila (2017), Altun, Xu, Inalhan, Vidal-Franco, & Hardt (2022). For the application of the proposed AI CM, the requirements were synthesized and defined at the architecture and each module level. They are as follows:

1. CM architecture requirements

- CM shall be a deterministic and finite process.
- CM identification shall identify the hazard scenarios unambiguously.
- CM shall be implemented to run onboard and be a highly automated process.
- CM shall be a reiterating process.
- The adequacy of the contingency procedures shall be ensured through flight tests and simulations.

Safety Monitor and CM Logic requirements

- The Safety Monitor shall continuously evaluate the sensor behavior by monitoring validated sensor data, or system behavior or any combination.
- The Safety Monitor shall detect behaviors that indicate that the system will leave the Safe State.
- When the Safety Monitor thresholds are exceeded, the CM Decision Logic shall select and activate a suitable Verified Function to ensure risk mitigation.

2. Contingency Planner requirements

- CM and analysis shall be conducted with the most current data.
- CM assessment methods shall calculate the ground risk of the area currently overflown.
- CM assessment shall calculate the air risk considering separation with other aircrafts.
- CM action should include the identification of alternative landing sites.
- CM mitigation and resolution strategies shall be approved by the competent authority.
- CM strategies shall calculate the reachability of landing sites.
- CM shall be defined within the strategic phase of the operation.
- CM shall be cooperative with other vehicles in the airspace.
- CM shall reduce flight time for aircraft experiencing the contingency.
- CM plan shall provide an adequate level of safety.
- CM plan shall be predictable to ATC and other airspace users.

A succinct summary comparing AI RTA and CM is presented in Table 18.

Table 18. Comparative description of AI RTA and CM

Category	AI Run Time Assurance	Contingency Management
Primary objective	RTA methods mitigates the risk of complex function misbehavior by managing the systems use of the AI/ML function output.	CM techniques mitigate risks in critical or off-nominal states and events to preserve safety of the aircraft.
Scope of concern	The AI/ML Function may be any function, algorithm, component or system that has not been subjected to aircraft assurance practices.	States: equipment failure (propulsion system failure, navigation sensors failure, GPS failure, flight controller failure, battery failure, control surface failure, lack of fuel, loss of communication with GCS etc. Events: weather conditions, obstacles, terrain, adversarial aerial vehicle, bird strike etc.
Architectural Modules	AI/ML Function Input Manager Safety Monitor Verified Function RTA Decision Logic RTA Switch	Input Manager CPG Safety Monitor CES CDL

3.1.4 Integrated AI RTA and CM

The conceptual architectures for AI RTA and CM are integrated into a unified AI/ML risk mitigation strategy, shown in Figure 51. The AI RTA is connected to CM via two links, namely AI RTA O/P and the input validation data. AI RTA O/P is the output obtained from either the AI/ML function or the verified function. CM input manager validates the data obtained from RTA and provides the downstream modules with validated data.

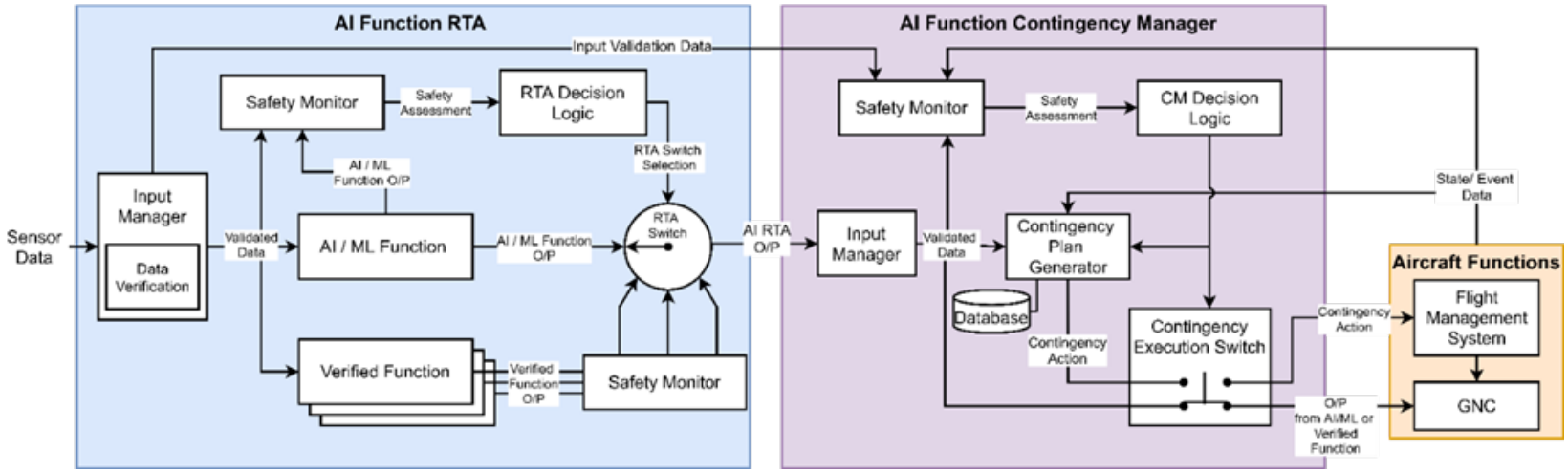

Figure 51. Functional architecture integrating RTA and CM

The second link connecting AI RTA and CM is the input validation data, which informs the safety monitor of CM about sensor integrity. Based on the sensor integrity, states, events, and validated data, the CM safety monitor checks safety assessment information to determine the risk. The safety assessment information is fed into the CM decision logic to take appropriate actions. The CM communicates with the aircraft functions by either providing a contingency action compensating for the failure of AI/ML function or simply forwarding the O/P of AI RTA.

3.1.5 Case study: detect and avoid with ML-enabled system

A conceptual case study to demonstrate feasibility of the AI/ML risk mitigation strategy has been conducted, based on a detect and avoid (DAA) system onboard aircraft. DAA systems are critical for UASm as they provide a solution to collision avoidance with intruders such as aircraft, building, power lines, etc. There are two main tasks of the DAA: detection of intruders and planning an avoidance maneuver. In this case study, intruder detection is performed using an electro-optical (EO) perception system (Xiang & Zhang, 2015; Tang, Lao, & Wan, 2022). Images from the perception module are fed into the convolution neural network (CNN) YOLOv4 (Bochovskiy, Wang, & Liao, 2020) to classify and determine location of the intruder. For a CNN-based vision perception system to be installed onboard an aircraft, there exist many challenges in certification (Vidot, Gabreau, Ober, & Ober, 2021; Boniol, et al., 2020), and risk mitigation is imperative for its safe integration into the aircraft.

The AI RTA and CM unified architecture for risk mitigation proposed in the previous section is adapted for the DAA case study, as shown in Figure 52. AI RTA is wrapped around the YOLOv4 function. The input manager validates the data from the EO sensor by checking the range of the pixels and for any inconsistencies. The downstream modules receive the validated data.

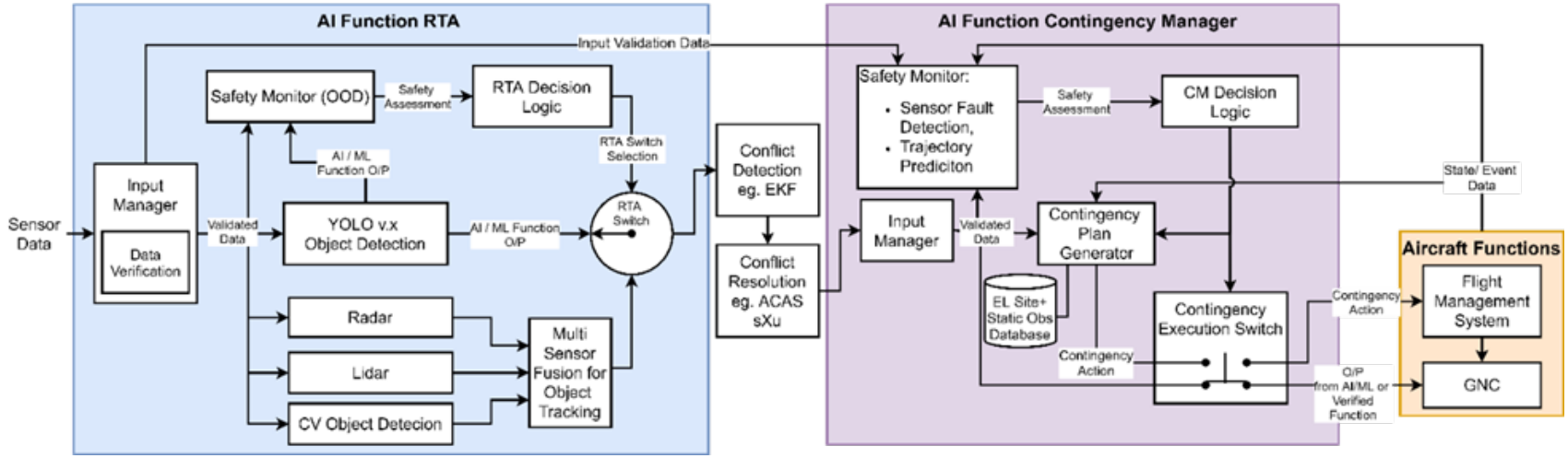

Figure 52. Integrated RTA and CM for DAA application with perception sensors

Verified object tracking function in this case study is based on a multi-sensor fusion system using radar, LIDAR, and computer vision, similar to methods proposed by Ramasamy, Sabatini, & Gardi (2014) and Dolph, Glaab, Allen, Consiglio, & Iftekharuddin (2019). The object tracking function uses a Gaussian filter to smoothly track the object and reduce the noise.

The safety monitor can be a collection of monitors which check different elements of a neural network (Rahman, Corke, & Dayoub, 2021) and score the output safety. An approach to monitoring a neural network is to detect if the image is OOD, e.g. an autoencoder-based OOD detector (Richter & Roy, 2017). The internal layer activation of the neural networks can also be used as a monitoring method, e.g., a saliency map-based failure prediction network (Mohseni, Jagadeesh, & Wang, 2019). Lastly, the output from the neural network can also be checked for its prediction quality or confidence and serve as a monitor, e.g., Deep Ensemble-based predictive UQ (Lakshminarayanan, Pritzel, & Blundell, 2017). In this case study, the neural network checks OOD data and provides UQ of neural network predictions.

Safety assessment information is provided to the RTA decision logic. Based on the assessment, the RTA decision logic instructs the RTA switch to select the safe output, either from YOLOv4 or a multi-sensor fusion method. An example of safety assessment is identifying the input image as OOD based on a set threshold, and then the decision logic instructs the RTA switch to select a multi-sensor fusion method as the output from the AI RTA. However, if the data is ID, the decision logic instructs the RTA switch to select YOLOv4 output.

The output from the AI RTA is fed to a filter that predicts if the aircraft is on a collision course with the intruder, and determines the conflict resolution strategy, such as ACAS sXu (Alvarez, Jessen, Owen, Silbermann, & Wood, 2019). This conflict avoidance maneuver, along with input validation data, act as input to the contingency manager. The input manager in the contingency manager again checks the data for validity, e.g. if the suggested conflict avoidance maneuver is within capabilities of the aircraft. Using this validated data from the CM input manager and AI RTA input manager, the safety monitor in the contingency manager assesses the current safety of the aircraft. The safety monitor checks multiple safety properties from component to aircraft level. At the component level, the safety monitor checks if there is any fault in the EO sensor, and at the aircraft level, it checks if trajectory predictions will lead to conflict based on current state and event information.

The CDL utilizes the safety assessment information to determine if there is a contingency scenario. In a scenario where the safety monitor deems there is a sensor fault, the CDL informs the CPG to initiate a conflict avoidance plan by changing the altitude and accounting for static

obstacle data from the database. This new plan is selected by the CES and sent to the flight management system for execution.

3.1.6 Conclusion

In this section, various existing RTA and CM architectures have been reviewed. Based on the review and weakness of existing approaches, a novel AI/ML risk mitigation approach based on a unified RTA and CM is proposed. Utilizing this unified architecture, a conceptual case study was conducted to show the feasibility of the unified RTA and CM architecture. Based on the case study, it was identified that existing methods still have challenges in monitoring perception systems and require further investigation.

3.2 Operational design domain-based risk mitigation of ML-enabled systems

The term risk is defined as the potential of an occurrence to cause harm defined by its probability and the severity of its consequence(s) (SAE, 2023). Due to the novelty of AI/ML-enabled systems, the risks associated with their deployment are not well understood. There is nearly always a non-trivial gap in transferring a model or algorithm from a simulation testbed to the real world. Being diligent about risk identification and mitigation strategies serves as a useful mechanism to increase system reliability. Risk mitigation approaches aim to reduce risk consequence throughout the lifecycle of the system. In a separate internal report submitted to the FAA (unpublished), risk mitigation for AI/ML-enabled systems using RTA and CM was reported along with the importance of the approach for risk mitigation based on ODD.

3.2.1 Operational design domain definitions

Note: The FAA uses the *ODD* initialism for *operational data domain.* However, due to limited material available from the FAA during research on this topic, the usage from available standards, literature, and regulators was adopted.

With the advent of autonomous vehicles (AVs), the drivers are partially or fully replaced with automation. This necessitates understanding and defining the operational boundaries for AVs within which they perform their intended function. This led to defining the term ODD. As per SAE (2021), ODD is defined as the

> operating conditions under which a given driving automation system or feature thereof is specifically designed to function, including, but not limited to, environmental,

> geographical, and time-of-day restrictions, and/or the requisite presence or absence of certain traffic or roadway characteristics. (SAE, 2021)

A recent publication from the ISO (2023) adds the safety dimension to the ODD definition by stating ODD defines "operating conditions under which a driving automation system is designed to operate safely."

A similar definition of ODD is adopted in the aviation domain, as published by EASA (2024). The ODD is defined as the "operating conditions under which a given AI/ML constituent is specifically designed to function as intended, including but not limited to environmental, geographical, and/or time-of-day restrictions" (EASA, 2024, p. 246). This definition narrows the scope of ODD to just the ML constituent. ODD for an ML model or constituent is also defined as the abstract description of its respective input space, as captured in its requirements, for which its behavior is (to be) designed (Agogino, et al., 2024). More generally, an ODD is the allocated portion of a specification of the intended operating environment(s) of a given system (including all foreseeable operating conditions) for which that system is designed and in which it is expected to fulfill its missions.

In aviation, the industry context of the operational environment and conditions are considered when writing requirements or performing system safety assessments that focus on system purpose and behavior. Defining the ODD provides an all-encompassing description of operational conditions and additional rigor to the development process for ML-enabled systems. The parameterization of ODD also constrains the problem space for assessing the performance of ML-enabled systems.

3.2.1.1 ODD characterization

The ODD can be broadly characterized into three domains: scenery elements, environmental elements, and dynamic elements, as shown in Figure 53 (ISO 34530, 2023). *Scenery elements* consist of the spatially fixed elements of the operating environment (e.g. roads, traffic lights, etc.) relative to the *ego vehicle*, the specific vehicle whose sensors and control systems are under consideration. *Environmental elements* consist of weather, atmospheric conditions, and connectivity of the vehicle. Lastly, *dynamic elements* consist of the movable items in the ODD, such as traffic, pedestrians, etc. If these broad characterizations do not fully capture the operating environment, measurements which reflect temporal elements are also considered.

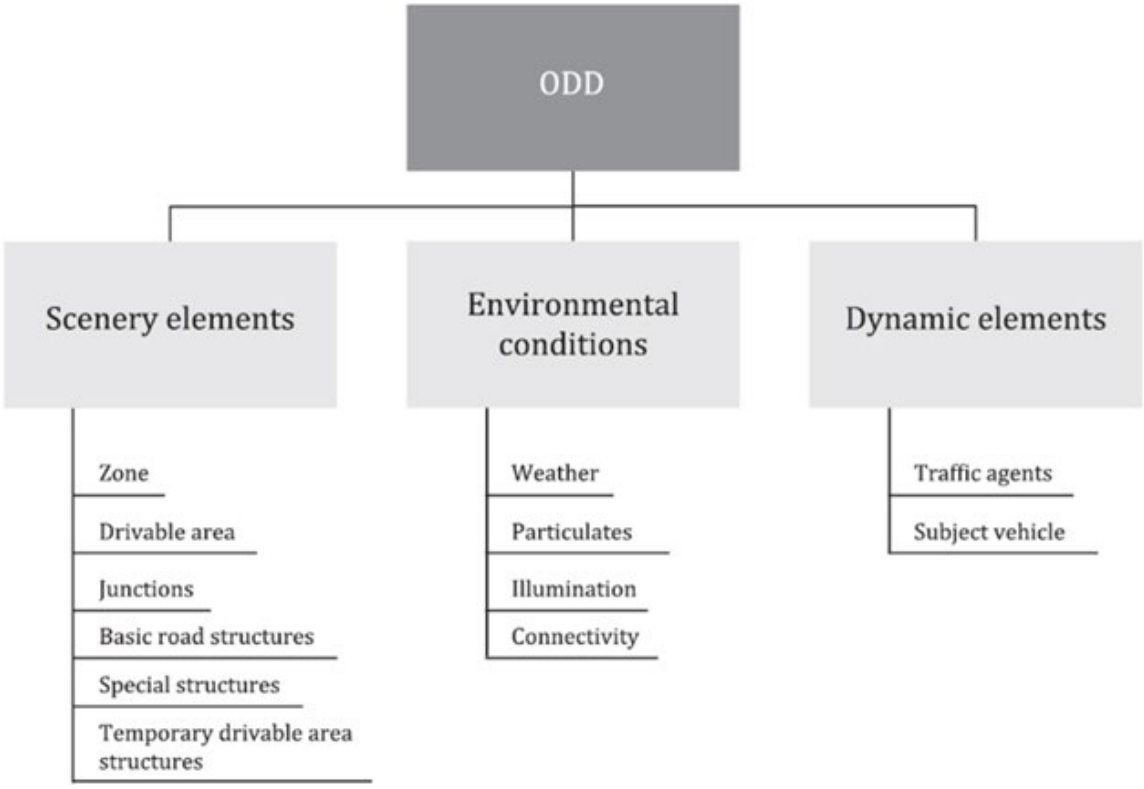

Figure 53. ODD characterization

When defining the ODD for a ML-enabled system, additional attention needs to be given to the type of data used for training. (Kaakai F. , Adibhatla, Pai, & Escorihuela, 2023) proposed as a data-centric approach to defining ODDs. Unlike traditional scenario-based ODDs, this approach categorizes data that ML models may encounter during operation into nominal, edge cases, and outliers, ensuring that ML systems are trained and tested with the right data. This method emphasizes the importance of characterizing the ODD during both the design and deployment phases to ensure robust safety performance.

3.2.1.2 Methods to define ODD

This section presents various approaches to define the ODD. The approaches are broadly classified into three categories: conceptual design framework, risk-based, and learning-based. There are very limited published approaches to define ODD in aviation compared to automotive. The approaches presented in this section are based on literature available in the automotive domain and can also help inform the aviation domain.

3.2.1.2.1 Conceptual design framework

The framework for designing an ODD, as outlined in the SAE Best Practice document (SAE, 2020), involves several key steps to ensure that automated driving systems (ADSs) operate within defined parameters. This framework is particularly relevant for SAE Level 4 ADS-operated vehicles and is designed to help developers, manufacturers, and regulators communicate and define the conditions under which ADS can safely operate. The main components of the framework are as follows:

1. *Identifying the Road/Route Network:* Developers must first map and identify the road or route network where ADS will operate. This step focuses on geographic boundaries and road characteristics relevant to the use case of ADS.

2. *Characterizing the Fixed Route Network and Infrastructure:* Once the route network is identified, the fixed infrastructure of the roads must be characterized. This includes physical features like traffic control devices, roadway geometries, and infrastructure (e.g., bridges, tunnels). This step ensures that the system is designed to handle the specific physical conditions of the deployment area.

3. *Identifying Operational Constraints:* Developers must test ADS on the identified network to discover operational constraints, such as weather conditions (e.g. rain, fog, snow), road surface conditions (e.g. cracks, potholes, snow-covered roads), or other dynamic factors that could impact ADS performance.

4. *Formulating an ODD Narrative:* Finally, a descriptive narrative is formulated that explains both the permissive (conditions under which ADS can operate) and non-permissive (conditions under which ADS cannot operate) elements of the system. This narrative is meant to communicate the system's limitations and capabilities to stakeholders, including regulators and the public.

Such procedural methods provide a good framework for defining the ODD. However, the challenge with such procedural methods is the difficulty in capturing the interrelated risks resulting from various ODD parameters.

3.2.1.2.2 Risk-based methods

Risk-based methods provide a systematic and principled approach to defining the ODD by assessing the likelihood and severity of potential hazards that may arise when a driving automation system operates under various conditions. The key factor in risk-based methods to define the ODD is the risk metric formulation.

The approach proposed by Hongseok (2020) develops a risk-based framework to define the ODD for driving automation systems. The framework identifies key conditions in a hyperdimensional space that influences system reliability and uses risk assessments to set the boundaries of acceptable operational conditions. The risk assessment framework first identifies the hazards, and a causal chain of events leading to and resulting from the hazard is established. Relevant conditions such as weather, road type, and lighting conditions are modeled to assess how they affect the probability of a system failure and the severity of its consequences. A risk matrix, like those used in aviation, is then applied to determine acceptable versus unacceptable conditions. Based on this analysis, the framework defines the boundaries of the ODD by combining the acceptable conditions across all hazards, ensuring the automation system only operates in low-risk environments.

Similar risk-based assessments are proposed by Lee, Nayeer, Garcia, Agrawal, & Liu (2020) based on a statistical risk assessment. The system evaluates each road segment and environmental condition by calculating the risk through severity, exposure, and likelihood of failure, i.e. $Risk = \sum_i (severity)_i \times (exposure)_i \times (likelihood)_i$, where i denotes the condition. By comparing these results to a predefined risk threshold, unsafe areas are excluded from the ODD. The innovation lies in quantifying the risk for geographic and environmental conditions to produce an ODD that minimizes operational risks.

A critical challenge when applying the risk-based methods for defining ODD is the challenge of determining the likelihood of failure for ML systems. Surrogate conditions are recommended in the literature to account for lack of direct data regarding certain conditions; however, its effectiveness needs to be further assessed. Lastly, the ODD defined in these approaches are at vehicle level, and mapping of these methods to the component level (i.e. to ML models) is challenging (Adedjouma, et al., 2024).

3.2.1.2.3 Learning-based methods

Alsayed, Resende, & Bradai (2021) present a method for monitoring the ODD of two-dimensional laser-based localization algorithms used in ADSs. The proposed system operates before the localization algorithm, using a grid-based feature extraction method called the "umbrella feature" to analyze laser scan data and determine if the environment is suitable for accurate localization. By employing ML classifiers, the system dynamically verifies whether environmental conditions are met for reliable operation. The method was tested in a simulated environment with various scenarios, achieving a balanced accuracy of 85.18% and high predictive performance. The approach enhances the safety of ADS by preemptively detecting error-prone scenarios, though further refinement is needed to reduce false alarms.

In some approaches, the ODD is not explicitly defined and is instead used directly in the training process to improve robustness of the learned model. Sun, et al. (2024) presents one such approach in addressing the challenge of enhancing perception systems in autonomous vehicles to function reliably beyond their predefined ODD, especially under adverse environmental conditions like unaccounted weather changes. The authors propose a robust learning framework that leverages perturbation models to simulate domain shifts and train perception models using two methods: robust model-based augmentation and worst-case learning. Results confirm that this framework successfully extends the ODD by making perception systems more robust to environmental changes.

Various ML models such as neural networks, random forest, stacking classifier, etc. have been proposed for learning the ODD. Such models take in as an input operational and environmental conditions and predict if the system is within ODD. The challenge with some learning-based methods is that they are treated as black box, and the explicit parameterization of the ODD is not well understood. As a result, it raises the question of completeness in the defining the ODD.

3.2.1.3 Aviation domain examples for ODD

This section provides ODD examples in the aviation domain and defines the specific conditions under which AI-based systems are designed to operate safely. In a vision-based landing system, the ODD includes the landing approach geometry, weather conditions, runway markings, and other environmental factors like visibility (Cappi, et al., 2024). This system's ODD is refined iteratively, incorporating both synthetic and real data to ensure compliance with operational requirements. For autonomous aerial refueling, the ODD encompasses conditions such as platform speed, turbulence, and the position of the refueling drogue, all critical for the system's ability to perform object detection and tracking using a DNN (Clement, Mottino, & Costello, 2024). Similarly, in collision avoidance systems, the ODD is derived from the Concept of Operations (ConOps), outlining environmental conditions like weather, geographical settings, and operational limitations such as airspace congestion (Stefani, Girija, Mut, Hallerbach, & Krüger, 2023). These factors define the boundaries within which the AI-based system must function, and scenario-based virtual testing is employed to validate the system’s performance against the ODD.

3.2.2 Risk mitigation based on ODD

As stated in the previous section, ODD is a set of measurable parameterizations of the operating conditions in which the system is designed to perform its intended function. Techniques have emerged that monitor the conditions under which the system is operating and identify departures

from the ODD. Any such departure poses a risk for ML-enabled systems to perform their intended functions. ODD departure information from monitors can inform decision making modules to execute appropriate risk mitigation steps. This section presents a review of different ODD monitoring techniques integral to risk mitigation.

Expected attributes for ODD monitoring are as follows (ISO 34530, 2023; ASTM, 2022):

- Monitor will be aware of real-time attributes of the ODD so that it can compare the external conditions to the predefined ODD values.
- Monitor should continuously evaluate the system and provide information to decide triggering of a risk mitigation maneuver.
- Monitor should be able to forecast and detect an upcoming ODD boundary with a sufficient time buffer so that the bounds are not violated.
- ODD should be understandable to the user in order to enable identification of ODD limits.

3.2.2.1 Learning-based monitors

Environmental conditions are an important element of the ODD. Approaches to detect the system operating environment are developed to identify if the system is operating in the desired weather conditions. Due to the complexity of detecting environmental conditions, using ML perception models for detecting environmental conditions from images is a popular choice (Introvigne, Ramazzina, Walz, Scheuble, & Bijelic, 2024; Triva, Grbić, Vranješ, & Teslić, 2022; Tanay Kondapalli, Vaibhav, Konda, Praveen, & Kondoju, 2021). One such approach describes the development of RECNet, a deep learning model designed to classify environmental conditions such as weather and road states in real-time using single RGB (red, green, and blue) frames (Introvigne, Ramazzina, Walz, Scheuble, & Bijelic, 2024). To address the challenge of operating autonomous vehicles in adverse conditions, the authors enhance the DENSE dataset by introducing a new label hierarchy and a semi-automated labeling pipeline that combines RGB and LIDAR data. RECNet, built on the EfficientNet-B2 architecture, achieves a 16% improvement in F1-Score over baseline models while maintaining real-time performance at 20 Hz. A challenge in such an approach is the complexity in labelling data. Creating detailed and accurate labels for complex weather conditions is difficult, particularly when multiple weather phenomena occur simultaneously.

The approach by Torfah, Xie, Junges, Vazquez-Chanlatte, & Seshia (2022) proposes a framework to enhance the safety of autonomous systems by learning runtime monitors that

predict when the system is about to exit the ODD. This approach uses counterexample-guided refinement and conformance testing to create predictive monitors based on observable system features. It provides the dual functionality of learning the ODD space and its boundary and of serving as an effective monitor. Evaluated in the context of autonomous driving, this framework successfully learns and monitors for a neural network-based lane-keeping module using the VerifAI toolkit and CARLA simulator. Results show that incorporating dynamic features like road location significantly improves the monitor's ability to prevent lane invasions. The paper claims that this method provides a systematic way to ensure safe operation of black-box ML components in safety-critical autonomous systems.

3.2.2.2 Operating parameter-based monitors

In some applications, directly monitoring the ODD might not be feasible due to its high dimensionality. For example, in computer vision systems the input is an image typically defined by RGB pixel values. In such systems, alternative parameters are measured instead of monitoring the ODD attributes directly.

Torens, et al. (2023) address the challenge of ensuring the safety and reliability of ML components in UAM, specifically in aviation systems using ML-based object detection. The authors propose an architecture that integrates runtime monitoring of the ODD to filter inputs based on conditions like altitude, geofencing, and image quality, ensuring that the ML component operates only within safe and valid parameters. This approach was tested on unmanned aircraft during flight trials using a human detection algorithm based on YOLOv4-tiny for landing area assessments. Results showed that filtering inputs within the ODD significantly improved detection accuracy, with precision and recall increasing when inputs outside the ODD were excluded.

OOD techniques are also used as alternatives to monitoring the ODD. Methods such as ODIN (Liang, Li, & Srikant, 2018) and UQ (Balduzzi, et al., 2021) are used to detect OOD inputs. For OOD images, neural networks often produce overconfident predictions, even when the input is irrelevant or unrecognizable. The ODIN method uses two techniques: temperature scaling to adjust the softmax output, thereby increasing the distinction between ID and OOD data; and input perturbations to amplify these differences.

3.2.3 Conclusion

In conclusion, process-oriented and learning-based frameworks exist for determining ODDs, though significant challenges arise when working in high-dimensional spaces, such as perception-based applications. In these contexts, alternative observable variables often need to be

defined to effectively capture the complexities of the environment. Another key issue is determining the ODD boundary and its associated buffer region, which remains a difficult task due to the inherent variability and uncertainty in these domains.

Additionally, measuring various ODD elements presents its own challenges. While there are approaches that utilize surrogate information, these methods are still in their early stages, and a more mature understanding of their effectiveness is needed. As the field continues to evolve, further research and development are necessary to address these limitations and enhance the reliability of ODD determination methods.Recommendations for Assurance Criteria, Future Standards, Policy and Certification

3.3 Human factor considerations for AI/ML-enabled systems

AI and ML could potentially automate a wide variety of aviation applications due to their ability to ingest large amount of information and provide complex decisions in a short time. Flight-deck automation is of particular interest, which might utilize AI and ML to address complex challenges related to safety, situational awareness, and confidence. By focusing on human-autonomy teaming and leveraging insights from relevant research streams, we can develop effective regulatory frameworks and identify research pathways that ensure safe and efficient integration of these advanced AI/ML enabled systems into modern aviation.

3.3.1 Review of automation levels

This section provides a brief review of various automation levels as defined by regulatory guidance, standards, and academic literature.

The concept paper published by EASA (2024) defines the classification of AI applications based on the degree of automation in the AI-based system:

- Level 1: Human Augmentation. The AI-based system assists a human operator.
- Level 2: Human-AI Teaming. The AI-based system collaborates with a human operator.
- Level 3: Advanced Automation. The AI-based system performs decisions and actions, either supervised by a human operator or not.

Figure 54 highlights the concept of Human-AI teaming as proposed by EASA, where the Level 2 classification is further sub-divided into 2A and 2B, proposing the notion of Human-AI cooperation and Human-AI collaboration, respectively. Cooperation is a process in which the AI-based system works to help the human operator accomplish his or her own goal. Collaboration is

a process in which the human operator and the AI-based system work together to jointly achieve a predefined shared goal and solve problems through a co-constructive approach.

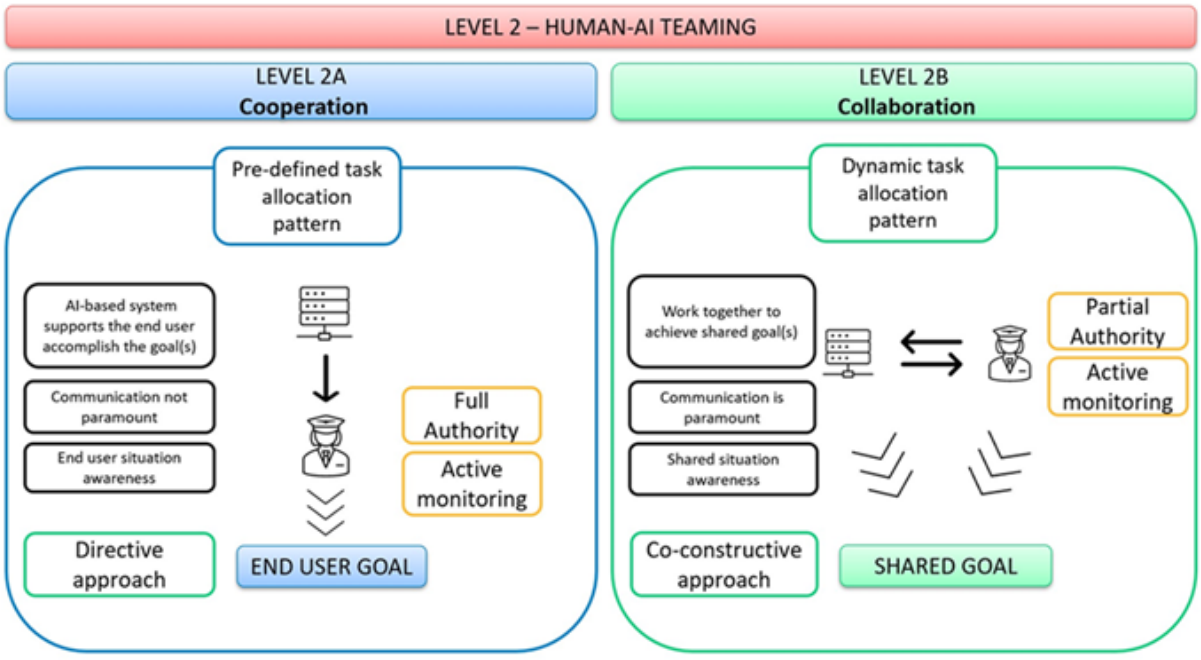

Figure 54. *Level 2 Human-AI Teaming as proposed in EASA Concept paper Issue 2*

SAE International has a structured framework for categorizing levels of driving automation in the automotive industry (SAE, 2021). This framework, comprising six levels of automation from Level 0: No Driver Automation to Level 5: Full Driving Automation, provides a clear progression towards fully automated driving capabilities. While the SAE levels do not explicitly emphasize the role of AI or ML in enabling autonomy, they offer a comprehensive breakdown of the evolution towards autonomous driving.

One of the earlier attempts to define levels of automation in comes from Parasuraman, Sheridan, & Wickens (2000), as shown in Figure 55, wherein they use a four-category 10-level system that is used frequently throughout academic research.

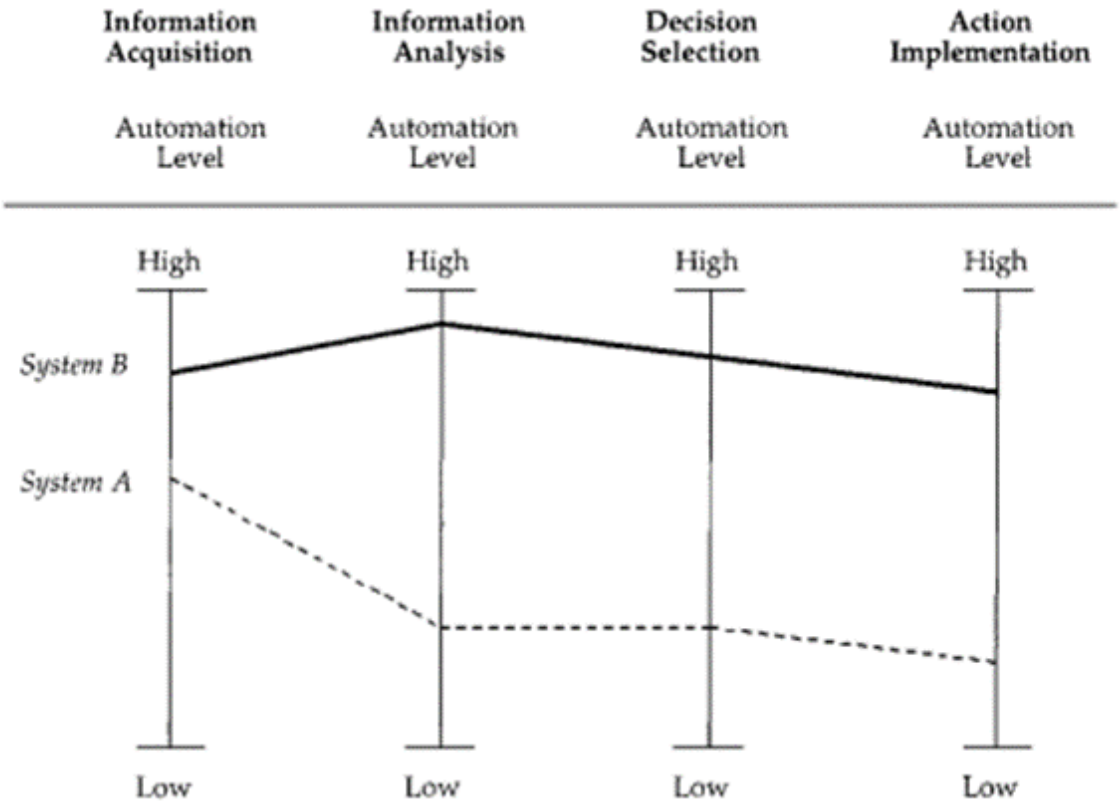

Figure 55. Levels of automation for categories of independent function

This subdivision allows for more thorough analysis and more effective problem-solving in many cases. For example, a system may be designed to have a high level of automation in terms of analysis and task selection, but a very low level of automation in action implementation. On a flight deck, such a system could decide upon a flight plan, present it, and even input many of its characteristics, but final confirmation and implementation would be left to the human users or other, more predictable systems.

It is crucial to comprehend the interaction dynamics between humans and autonomous systems, as this understanding is pivotal in guiding assurance activities. The recommendations put forth by the tasks described in this report are specifically aimed at EASA Level 2 autonomous applications. However, we are narrowing our focus to identify and address the challenges unique to Level 2 aviation systems, with a particular emphasis on the Human-Autonomy teaming aspect. By delving deeper into the intricacies of this relationship, we aim to enhance the effectiveness and seamless integration of human operators with autonomous technologies at this intermediate level of autonomy.

Regulating flight-deck automation at the EASA 2A/2B level presents several challenges:

- *Ensuring Safety:* Ensuring that automated systems operate safely under all conditions is paramount. This includes developing robust testing and certification processes.
- *Maintaining Situational Awareness*: Pilots must maintain situational awareness even when relying on automated systems. This requires designing interfaces that provide clear and timely information.
- *Justified Confidence*: Pilots need to have confidence in the automated systems to use them effectively.

Several research streams are relevant to addressing these challenges:

- *System Transparency:* Research on making AI systems more transparent to users, including methods for providing system transparency and understanding AI behavior.
- *Trust Models:* Studies on developing appropriate trust relationships between humans and AI, including trust predictors and methods for capturing behavioral evidence of trust.
- *Situational Awareness*: Research on maintaining and regaining situational awareness in high-level automation environments, including methods for information integration and overcoming adversarial attacks.
- *Individual Differences*: Investigations into how individual differences, such as cognitive biases and personality components, influence interactions with AI systems and perceived risks.

Lastly, although the levels of automation reviewed are not directly transferable to aviation, they offer valuable insights for analyzing autonomous systems alongside EASA's definition. By examining the different levels of automation and the corresponding human-machine interactions outlined, stakeholders can gain a deeper understanding of the challenges and advancements in autonomous technology deployment.

3.3.2 Applications involving Human-AI interactions in aviation

This section explores specific applications within aviation, focusing on collaborative tasks where both human and machine actions are essential. In aviation, Human-AI teaming at the 2A/2B AI level is critical for ensuring safe and efficient operations. These applications often involve dynamic task allocation and shared situational awareness between pilots and AI systems.

3.3.2.1 Enhanced flight path management

AI-based systems assist pilots by suggesting optimal flight paths based on real-time data, including weather conditions, air traffic, and aircraft performance (Rosenhow, Lee, & Li, 2022). The system and the pilot work collaboratively to adjust the flight path as conditions change, ensuring safety and efficiency.

3.3.2.2 Foreign object debris detection

Human-AI teaming in foreign object debris (FOD) detection applications enhances the ability to monitor and maintain safety in aviation manufacturing environments. AI-based systems can analyze video feeds from factory floor cameras to identify foreign object debris left in critical areas of an aircraft during construction and alert appropriate personnel. These systems work collaboratively with human operators to ensure a rapid and effective response, thereby minimizing the risk of damage to aircraft components and ensuring the highest standards of safety and quality.

3.3.2.3 Training

Human-AI teaming in aviation training can enhance the learning experience for pilots and other aviation professionals (Kabudi, Pappas, & Olsen, 2021). AI-based adaptive learning platforms provide personalized training programs for aviation students, adjusting content and difficulty based on individual performance and learning styles. These platforms work collaboratively with human instructors to ensure effective learning outcomes.

3.3.2.4 Design

In aviation design, Human-AI teaming can enhance the development of aircraft and related systems (Koch, 2017). AI-based design tools assist engineers in creating and optimizing aircraft components. These tools analyze design parameters and provide recommendations, while human designers make final decisions and adjustments.

3.3.2.5 Analysis

Human-AI teaming in aviation analysis can enhance the ability to process and interpret large datasets, improving decision-making and operational efficiency. AI-based systems analyze data from aircraft sensors to predict maintenance needs and identify potential issues before they become critical. These systems work collaboratively with human maintenance personnel to ensure timely and effective maintenance.

3.3.2.6 Autonomous ground operations

While primarily focused on ground transportation, automobile applications of Human-AI teaming can inform aviation practices, particularly in ground operations and logistics.

Autonomous ground vehicles are used in airports for tasks such as baggage handling and aircraft towing (Franke, Garrett, Pasareanu, Malik, & Hemann, 2015). These vehicles operate under the supervision of human operators, who can intervene and provide guidance as needed.

3.3.3 Challenges in Human-AI autonomy

AI can be used for different functions to enable autonomy, including for object recognition, decision making, control loop execution, etc. AI may not be standalone, but embedded into an autonomy pipeline. The integration of such AI-enabled systems into flight-deck automation at the 2A/2B AI level and their associated functional criticality presents several unique challenges (National Academies of Sciences, Engineering, and Medicine, 2022). These challenges are broadly classified as system and user characteristics. This section provides a high-level summary of these challenges.

3.3.3.1 System characteristics

3.3.3.1.1 Predicting and understanding the behavior of AI-based systems

One of the primary challenges in Human-AI autonomy is predicting and understanding the behavior of AI-based systems. This is crucial for ensuring that pilots can effectively interact with and rely on these systems.

- **System Transparency:** Providing system transparency is essential for users to understand AI behavior. Research in explainable AI (XAI) has shown that transparency can significantly improve user trust and system usability. Methods for providing system transparency include visualizations, natural language explanations, and interactive interfaces that allow users to query the AI system.
    - **Shared Situation Awareness**: Effective Human-Autonomy teaming requires shared situation awareness. This involves designing displays and information systems that provide a comprehensive view of the current state of the system and the environment.
    - **Emergent Behaviors**: Predicting emergent behaviors in human-AI team interactions is another significant challenge. These behaviors can arise from the complex interplay between human actions and AI-based system responses. Developing predictive models of human-AI team performance, both in routine and failure conditions, can help in anticipating and mitigating potential issues. However, developing such models is difficult.

3.3.3.1.2 Development of an appropriate trust model

Ensuring the trust (also known as *justified confidence* in aviation) of AI-based systems in flight-deck automation is critical for safety and effectiveness. Kaplan, Kessler, Brill, & Hancock (2023) present meta-analysis to determine factors that help predict trust in AI-based systems, as shown in Figure 56. However, existing human-AI research is limited in terms of conceptualizing functions, metrics, and performance-process outcomes for dynamically evolving, distributed, and adaptive collaborative tasks.

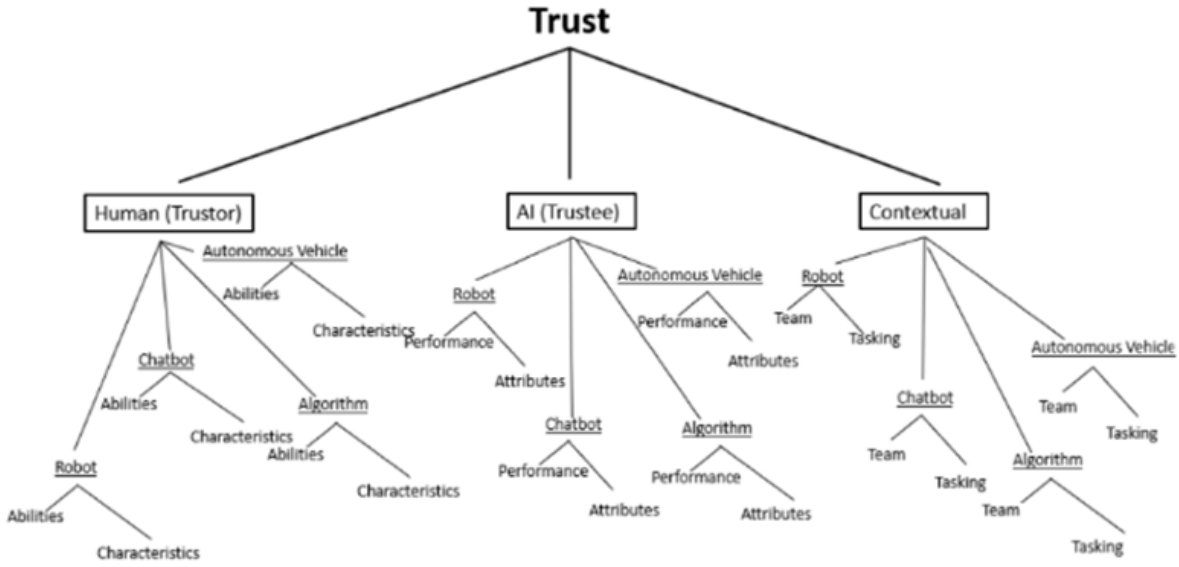

Figure 56. A tree-based model on factors influencing trust in AI-based system

- **Trust Calibration**: Trust calibration models are essential for developing appropriate reliability models. These models aim to calibrate the level of trust users have in AI-based systems to ensure it is appropriate for the system's capabilities and limitations. Factors such as system transparency, reliability, and user experience play a crucial role in trust calibration.
- **Behavioral Evidence of Trust**: Establishing methods for capturing behavioral evidence of trust is important for dynamic task allocation. This involves monitoring user interactions with the AI-based system to identify over-trust or under-trust events and adjusting the system's behavior accordingly.
- **Real-World Performance**: Many measures of team performance do not address the real-world performance demands of complex and dynamic multidomain operation tasks.

Developing metrics that capture the effectiveness of human-AI collaboration in real-world scenarios is essential for evaluating and improving system reliability.

- **Trust Predictor Capture**: Many factors can influence the trust levels of a user over time. These sources have been identified in a variety of contexts, but there is evidence that trust may have unique properties that emerge dependent upon the domain in question. Finding methods to predict and capture the potentially unique properties of trust formation within the flight deck domain will be a continuing challenge in ensuring quality human-automation teaming.

3.3.3.1.3 Automation strategies

Managing different levels of automation within flight-deck systems presents unique challenges, particularly in maintaining situational awareness and ensuring effective human-AI collaboration.

- **Adaptive Automation**: Adaptive automation involves systems that can adjust their level of automation based on the user's current state, task demands, and environmental conditions. This approach can help in maintaining situational awareness and reducing cognitive load.
- **Multiple AI-based Systems**: Managing multiple AI-based systems, each operating at different levels of automation, requires effective coordination and integration. Techniques such as multi-agent systems and distributed AI can facilitate the management of multiple AI-based systems, ensuring they work together seamlessly.
- **On-the-Loop Operations**: Methods for maintaining or regaining situational awareness during on-the-loop operations are crucial. This involves designing interfaces that provide clear and timely information, allowing users to monitor and intervene when necessary.

3.3.3.1.4 Situational awareness

A critical factor in ensuring the safety and effectiveness of flight-deck automation is situational awareness. Jiang, Karran, Coursaris, Léger, & Beringer (2022) recommend situational awareness as an important factor in understanding human-AI interaction. Maintaining and regaining situational awareness in high-level automation environments presents several challenges, including:

- **Information Integration**: Methods to support information integration, prioritization, and routing across multidomain operations need to be investigated. This involves designing systems that can effectively aggregate and present information from multiple sources.

- **Adversarial Attacks**: Developing methodologies for detecting and overcoming adversarial attacks on situational awareness is essential. This includes techniques for identifying and mitigating false or misleading information that could compromise situational awareness.
- **Self-Awareness**: The effectiveness of an AI-based system's self-awareness for improving human-AI coordination is another important consideration. Such self-aware systems can monitor their own performance and provide feedback to users, enhancing situational awareness and collaboration.
- **Synchronization of Belief State**: Synchronizing the belief state between human and AI-based systems for situational awareness is crucial. This involves ensuring that both the human and the system have a shared understanding of the current situation and can make informed decisions based on this shared understanding.

3.3.3.2 User characteristics

3.3.3.2.1 Individual differences in biases and perceived risk

Understanding individual differences in perceived risks and benefits is essential for designing effective human-AI systems. These differences can significantly influence how users interact with and trust AI-based systems.

- **Human and AI Biases:** Understanding the interdependencies between human and AI-based system bias is crucial. Research has shown that biases can emerge from AI learning based on small and sparse datasets, and these biases can influence user perceptions and interactions.
- **User Characteristics:** User differences influence the behavior of human-AI systems as well. A user's competency, expertise, culture, gender, and big five personality traits[1] all have significant effects upon user behavior and trust (Kaplan, Kessler, Brill, & Hancock, Trust in artificial intelligence: Meta-analytic findings., 2023).
- **Personality Components:** Personality components have been shown to influence how reliability information is interpreted. For example, individuals with high levels of trust

[1] Openness to experience: Includes intellectual curiosity and creative imagination
Conscientiousness: Includes organization, productivity, and responsibility
Extraversion: Includes sociability and assertiveness
Agreeableness: Includes compassion, respectfulness, and trust in others
Neuroticism: Includes tendencies toward anxiety and depression

propensity may be more likely to trust AI-based systems, while those with high levels of skepticism may require more evidence of reliability.

- **Perceived Risks and Benefits:** Determining human-AI biases that emerge from AI learning and understanding how these biases influence perceived risks and benefits is essential. This involves designing systems that can adapt to individual differences and provide personalized feedback and support.

3.3.3.3 Mitigation approaches

To address these challenges and ensure the safe and effective use of AI in aviation, various mitigation approaches can be employed. This section explores these approaches in detail, building upon situations and challenges identified in the previous section.

3.3.3.3.1 Human-centered design

Human-centered design (HCD) is a methodology that places the user at the center of the design process, ensuring that AI-based systems are intuitive, transparent, and aligned with human needs and capabilities. A HCD approach is proposed by Mathewson & Pilarski (2022) for the interactive ML field. Aspects of this approach are crucial for addressing challenges related to system transparency, situational awareness, and trust.

Within the context of automated systems, the first step in HCD is to clearly define the goals of the AI-based system. The process of precisely defining the goals of the system ensures that the system is properly aligned to the needs of the users of the system and allows for an early understanding of issues of bias or misalignment that may occur later. This requires understanding the specific needs and challenges of the users and ensuring that the system's objectives align with these needs, as users feel more at ease when system output matched their own judgement (Bach, Kristiansen, Babic, & Jacovi, 2024) .

Involving users throughout the design process is essential for ensuring that the system meets their needs and expectations. This can be achieved through iterative design processes that incorporate user feedback at relevant stages.

Designing AI-based systems involves making trade-offs between various factors, such as cost, storage, learning speed, inference speed, computational complexity, and human interpretability. These trade-offs must be carefully analyzed to ensure that the system is both effective and practical.

3.3.3.3.2 Model interpretability and explainable AI (xAI)

In interpretable ML, two important factors are the context and the end task related to human factors (Murphy K. P., 2023). When formulating an interpretable ML problem, it is crucial to define the context and the end task. These two elements determine the suitable metrics for evaluating the performance of the end task and also to indicate the important properties that should be considered in the explanation. Additionally, context plays a key role in identifying and selecting data and training metrics for the ML model. The choice of explanation methods depends on the desired model and properties and is evaluated based on the end-task metric to determine the downstream performance.

Explainability and interpretability involve communicating the AI-based system's rationale in a manner that users can understand. This includes identifying which rationale information should be communicated and how. Further explanation of system actions is more critical when there is misalignment in behavior expected by the user and exhibited by the system (Zhang, et al., 2024).

Providing transparency involves making the system's decision-making processes clear and understandable to users. This can be achieved through various methods, such as visualizations, natural language explanations, and interactive interfaces.

3.3.3.3.3 Trust calibration

Trust calibration is essential for developing appropriate trust relationships between humans and AI-based systems. This involves ensuring that users have the proper level of trust in the system's capabilities and limitations. Models on trust calibration aim to calibrate the level of trust users have in such systems. These models consider factors such as system transparency, reliability, and user experience.

The word “trust” has received some criticism in the aviation industry due to a number of concerns, most notably an unwillingness to anthropomorphize the machine by implying it is “trustworthy.” The use of “trust” in this report is in part because it grows out of the research on interpersonal trust (Mayer, Davis, & Schoorman, 1995), but also because the word itself means something different in this context. In trust research, the word most frequently refers to certain categories of behavior, such as complacency, skepticism, honesty, kindness, etc. Also, Hoff & Bashir (2015) point out notable differences between trust in human-autonomy and the interpersonal domain. A wealth of literature on trust in the human-autonomy field exists and can be leveraged for the aviation domain.

Capturing behavioral evidence of trust involves monitoring user interactions with the AI-based system to identify over-trust or under-trust events. Figure 57 is an example of how behavior

might be coded to capture behaviors responding to an alarm on a flight deck (Haskins, 2021). This diagram reconciles three different models for automation trust: use/misuse/disuse/abuse, in/appropriate reliance/compliance, and complacency/bias. Information on trust behaviors can be used to adjust the system's behavior and improve trust calibrations.

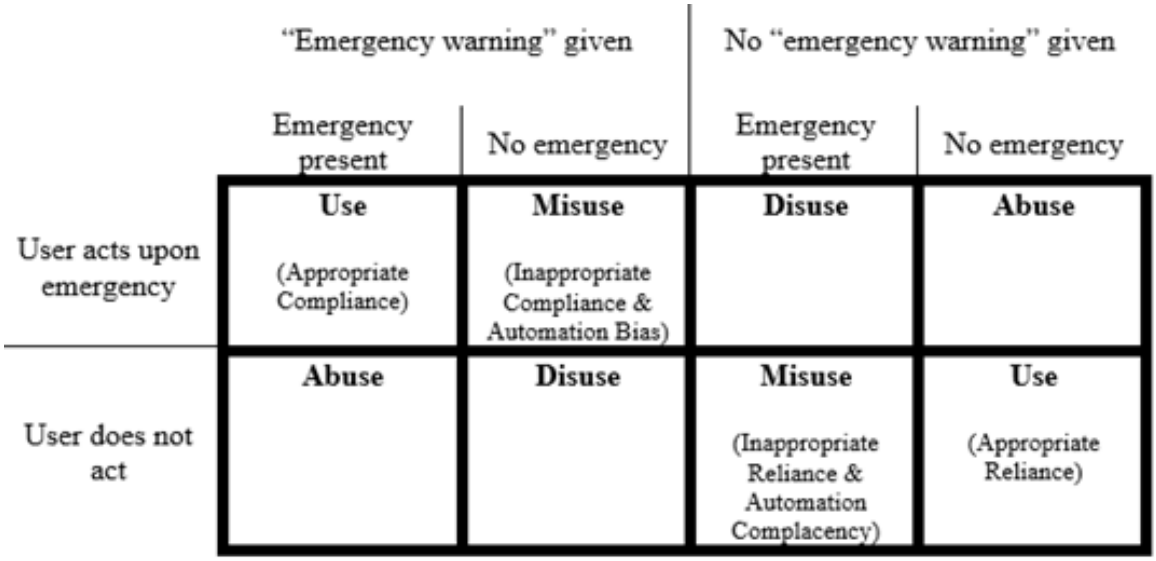

Figure 57. Reconciliation of automation error frameworks

3.3.3.3.4 Adaptive automation

Adaptive automation involves systems that can adjust their level of automation based on the user's current state, task demands, and environmental conditions. This approach is crucial for maintaining situational awareness and reducing cognitive load. This approach to system design usually relies on an internal trust model of the user to determine whether to increase or decrease user control of the system.

Dynamic task allocation involves real-time reallocation of tasks between humans and automation based on current conditions, performance, and situational demands. For example, in predictive maintenance, an AI-based system can dynamically adjust its recommendations based on the current state of the aircraft and the maintenance personnel's workload.

Context-aware systems use contextual information, such as environmental conditions and user state, to adapt their behavior and support more effective human-AI interactions. For instance, in enhanced flight path management, an AI-based system can adjust its flight path suggestions based on real-time weather conditions and air traffic.

3.3.4 Conclusion

This section provided an overview of various considerations for systems in which a human interacts with autonomy which is enabled by AI. Such systems can have AI-based functions deployed on a spectrum of low- to high-criticality tasks, such as decision making, control, and planning. As the level of automation increases along with associated criticality level of the systems, challenges in Human-AI interactions are observed, as noted in the literature. Although most of the literature reviewed for this section was not from the aviation domain due lack of extensive research on this topic in aviation, numerous important lessons can be adopted from other domains.

3.4 Introduction for recommendations

ML models pose a significant challenge in fully applying traditional aerospace certification standards and methods, as ML is data-driven while traditional aerospace standards tend to be requirements-driven. As a result, significant rethinking is required in implementing new or augmented standards and methods, which motivates the development of associated certification strategies.

This chapter recommends strategies for the guidance of AI/ML assurance development, standardization, certification strategies, future policy/guidance, and training. These recommendations are based on the findings of various efforts and a review of existing standards, academic surveys, and discussions with Boeing's AI SME and Organization Designation Authorizations (ODAs). These efforts include results obtained from assessments of AI/ML V&V technologies, implementation techniques, and exploration of safety risk mitigation.

This section first provides the scope of the recommendations, detailing the level of AI system considered and factors that will influence future evolution of the recommendation. A typical AI/ML model development process is presented to provide a deeper understanding of the associated workflow. This development process is broken into two phases: design time development and run-time development. Various metrics are used for evaluating the ML model during its development process. Having set the baseline understanding of ML model development, existing aerospace standards developed by RTCA and SAE are discussed and their shortcomings for use with AI are described. Following the section on existing standards, a preliminary list of recommendations is provided, with list of some potential techniques and tools. Lastly, the conclusion and future work is presented.

3.4.1 Scope

There exists a variety of ML algorithms and applications for aerospace systems (SAE, 2021). The recommendations were scoped to be more tractable in the given timeframe of the project. Level 3 Autonomy AI/ML applications as defined by EASA (EASA, 2023) are those where the AI/ML based system makes decisions or actions with minimal supervision from the human operator.

As the recommendations were being developed, traditional development assurance processes (SAE International, 2010; RTCA, Inc., 2011; RTCA, Inc., 2000) were used and augmented with additional requirements development and V&V approaches, methods, and tools. These augmentations will aid in aligning the traditional development process with the development of AI/ML systems.

The recommendations provided are non-exhaustive and expected to evolve as maturity is attained through Boeing's internal AI/ML V&V process development. Details of these approaches are to be captured and negotiated as means-of-compliance issue papers with the FAA to establish the metrics, tools, and methods. Boeing intends to show compliance to established regulations dealing with intended function and safety. Lastly, flight test experiments in real world scenarios will also be instrumental in developing and evolving existing and future recommendations for AI/ML assurance development and certification strategies.

3.4.2 Machine learning model development process

The AI/ML model development process consists of various stages. The first is the design time development process, which occurs in a benchtop-type environment. Following the design development is the run-time development process, which involves testing/evaluating the ML model in realistic environment.

3.4.2.1 Design and development considerations

The Design Time development process workflow consists of various steps, but an abbreviated version relevant for the construction of an ML constituent is shown in Figure 58. At a high level, the process consists of three phases: design, development, and deployment; each is color-coded in Figure 58. The process workflow illustrates the development of ML model from scratch, for example, defining the network architecture, activation function, convolution filter, number of nodes, etc. However, occasionally in the application of neural network variants used in computer vision applications like in YOLO variants, a transfer learning approach is employed. Transfer learning involves freezing part of a pretrained network and only training the last few layers based on the requirements.

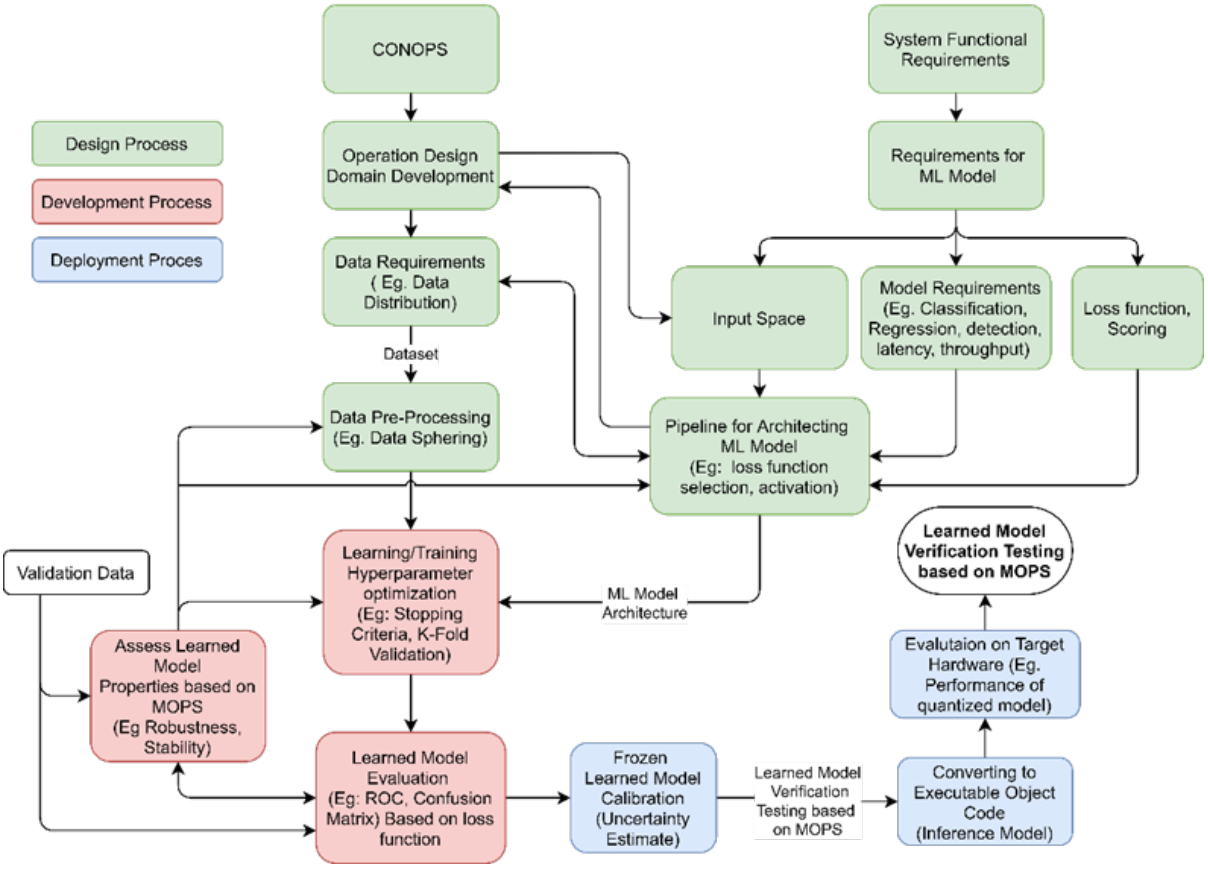

Figure 58. ML model development flow diagram

ConOps can be derived from high-level operational models and can be based on operational environment, performance criteria, standard operating procedures, contingency procedures, etc. Derived information from ConOps and aircraft function requirements define the ML constituent system requirements. The ODD is extracted from ConOps. The ODD is defined by the environment in which the ML constituent is designed to operate. At minimum, the ODD should include limits and boundaries of operation of the ML constituents within which it can operate safely.

Data requirements are critical for creating the dataset for ML constituents and are obtained from the ODD. Data requirements must cover all aspects of the ODD. Even if data requirements cover all aspects of the ODD, some additional considerations while curating the dataset include:

- Ensuring your dataset covers all aspects of ODD. For a perception model, this includes different lighting conditions, weather scenarios, object varieties, etc.

- Analyzing the frequency of each class or label in your dataset. A balanced distribution ensures that the model is not biased towards more frequent classes.
- Ensuring that all relevant scenarios (e.g., different traffic situations, ground crew interactions) are adequately represented.
- Ensuring the dataset includes data from different times (day, night, twilight) and various geographical locations if applicable.
- Identifying and quantifying rare but critical scenarios, which are often crucial for safety-critical applications, such as for perception models in vehicles.
- For models relying on sensor data, ensuring that the dataset includes data from all relevant types of sensors and that it represents the diversity of sensor outputs.

For additional information on these topics readers can review *Data-Centric Operational Design Domain Characterization for Machine Learning-Based Aeronautical Products* (Kaakai F. , Adibhatla, Pai, & Escorihuela, 2023) and *Data Validation for Machine Learning* (Breck, Polyzotis, Roy, Whang, & Zinkevich, 2019).

After data collection is the data preprocessing/feature engineering step. In this step, the data is normalized/standardized to a standard range or distribution to avoid bias towards certain features. Techniques like deletion for missing values are used for handling missing data, depending on the context. Additionally, artificially increasing the dataset size and diversity using techniques like rotation, scaling, and flipping of images are also employed.

Parallel to the dataset preparation step is defining the ML constituent model based on ML system requirements. This involves defining the category of ML model to be used (e.g. classification, regression, etc.) and defining the loss function (e.g. hinge loss, negative log likelihood, among many others (Ciampiconi, Elwood, Leonardi, Mohamed, & Rozza, 2023; Wang, Ma, & Zhao, 2020). All of this information combines to help define the pipeline for creating the ML model, where initial exploratory analysis is conducted to determine efficacy of the model.

Following the design process step is the development process, when the ML constituent training and evaluation occurs. The learning process involves defining the stopping criteria to prevent underfit/overfitting or understanding how well the model generalizes within the given dataset. Validation methods such as k-fold cross validation, stratified cross validation, and many more may be utilized. The stopping criteria can be also be defined by meta-algorithms (Goodfellow, Bengio, & Courville, 2016).

Most deep learning models have associated hyperparameters selected during the training process. A non-exhaustive list of hyperparameters is shown in Table 19. Hyperparameter tuning is very important, as hyperparameters affect the model resulting from the training process. Tuning is often considered a multi-objective optimization problem. There are various approaches to tuning hyperparameters including grid-based, random search, and Bayesian optimization (Feurer & Hutter, 2019). Grid-based methods involve defining a finite set of values for each parameter and evaluating the model on this set; however, it suffers from the curse of dimensionality. Random search methods sample hyperparameters randomly and are useful as a baseline, since they make no assumption about the model. However, population-based methods such as genetic algorithms and evolutionary algorithms outperform random or grid search. An alternative to these methods is Bayesian optimization, which uses a probabilistic surrogate model and an acquisition function. The acquisition function uses predictive distribution of the surrogate model to determine the utility of various candidate hyperparameter values.

Table 19. Non-exhuastive list of hyperparameters and their effect on models

Hyper-Parameter	**Change**	**Reason**	**Caveats**
Number of hidden Units	Increase	Increasing the number of hidden units increases the representational capacity of the model.	Increasing the number of hidden units increases both the time and memory cost of essentially every operation on the model.
Learning Rate	Tuned Optimally	An improper learning rate, whether too high or too low, results in a model with low effective capacity due to optimization failure.	
Convolution Kernel Width	Increased	Increasing the kernel width increases the number of parameters in the model.	A wider kernel results in a narrower output dimension, reducing model capacity unless you use implicit zero padding to reduce this effect. Wider kernels require more memory for parameter storage and increase runtime, but a narrower output reduces memory cost.
Implicit zero Padding	Increased	Adding implicit zeros before convolution keeps the representation size large.	It increases the computational and memory cost of the operation, because you are performing math on

Hyper-Parameter	Change	Reason	Caveats
			"empty" pixels (zeros) that contain no real information.
Weight Decay Coefficient	Decreased	Decreasing the weight decay coefficient frees the model parameters to become larger.	
Dropout Rate	Decreased	Dropping units less often gives the units more opportunities to "conspire" with each other to fit the training set.	

Once a candidate model or models are trained, they are evaluated for their performance. There exist various metrics to determine the performance of ML constituents, which differ based on types of model and assessment objective. A comprehensive list of metrics is provided by the ISO (2022) for assessing performance of classification model, as summarized in Table 20. For additional information, readers are directed to *Evaluating learning algorithms: a classification perspective* (Japkowicz & Shah, 2011) and *A Review of Evaluation Metrics in Machine Learning Algorithms* (Naidu, Zuva, & Sibanda, 2023).

Table 20. Metrics for assessing performance of classification models

Assessment Metrics	Binary Classification	Multi-Class Classification	Multi Label Classification
Performance Metrics	Confusion Matrix, Accuracy Precision, Recall, Specificity, F_1 Score, F_β, KL-Divergence, ROC, AUROC, Cumulative Response Curve, Lift Curve	Confusion, Accuracy, Macro/Weighted/Micro Average- Precision, Recall, Specificity, F_1, KL- Divergence	Hamming Loss, Exact match ratio, Jaccard Index, KL-Divergence
Computational Complexity	Latency, Throughput, Efficiency, Energy Consumption		

After evaluating the performance of the learned ML constituent, it is also important to test for various properties of the ML constituent, including robustness, generalization, and many more. *Robustness* is the ability of a system to maintain function and performance within the ODD specifications despite deviation from nominal and/or adverse conditions. To ensure a comprehensive evaluation of robustness, it is important to go beyond solely assessing the trained model. The robustness of the entire system, including the sub-functions that comprise the ML component, should also be taken into consideration.

The IEEE (2023) provides test specifications with a set of indicators for common corruption and adversarial attacks used to evaluate the robustness of an AI base image recognition service. Some examples of common image corruptions include noise, blur, weather, digital changes, and mask. Examples of common adversarial attacks include transferable adversarial attacks and query attacks. The IEEE (2023) proposes the following metrics for evaluation:

- *Accuracy:* The capability of the target service under evaluation to produce correct predictions when no corruptions or attacks are introduced.
- *Robustness against Corruption:* The average capability of the target service under evaluation to produce correct predictions when corruption is introduced.
- *Robustness against Adversarial Attacks:* The average capability of the target service under evaluation to produce correct predictions under an adversarial attack.
- *WorstCase Robustness against Corruption:* The average capability of the target service under evaluation to produce correct predictions when *all corruptions* being tested against are introduced.
- *WorstCase Robustness against Adversarial Attack:* The average capability of the target service under evaluation to produce correct predictions under *all adversarial* attacks being tested against.

In summary, while robustness focuses on the average capability of the model to handle corruptions and adversarial attacks, worst-case robustness considers the model's performance under the most challenging and severe scenarios, encompassing all possible corruptions and adversarial attacks being tested against.

The ISO (2021) provides additional metrics for testing robustness of neural networks in general, as shown in Table 21.

Table 21. Performance measures for robustness evaluation of neural networks

Statistical	Empirical	Formal
Interpolation/Regression: • Root Mean Square Error (RMSE) • Max Error • Actual/Predicted Correlation Classification: • True positive rate • True negative rate • Negative predictive value	• Field trials • A posteriori testing • Benchmarking of neural network	• Using uncertainty analysis to prove interpolation stability • Using solver to prove a maximum stable space property • Using optimization techniques to prove a maximum stable space property • Using abstract interpretation to prove a maximum stable space property

Once learned model is qualified via test and evaluation, it is frozen before being deployed on the target hardware, where it must be calibrated and uncertainty estimates of the system must be quantified. Methods to calibrate learned model include expected calibration error, temperature scaling average calibration (mean absolute calibration error, RMSE calibration error, miscalibration error) adversarial group calibration, sharpness, proper scoring rules (negative log likelihood, check score and accuracy (mean absolute error, median absolute error, correlation) (Chung, Char, Guo, Schneider, & Neiswanger, 2021). Uncertainty estimates of ML constituent outputs can be obtained employing techniques like deep ensemble, hyperparameter ensemble, Bayesian neural networks, etc. (Tran, Lakshminarayanan, & Snoek, 2020).

Lastly, once the model is ready to deploy on target hardware associated with target performance, requirements need to be considered and accounted for. This might include quantization of the model to reduce size, improve throughput/latency, etc., and might require re-evaluation.

3.4.2.2 Run-time considerations

After the design and development process is complete, deploying an ML model involves run-time considerations to ensure effectiveness and efficiency. Some key considerations are provided below.

The decisions and actions derived from the ML constituent should consider the cost of making incorrect predictions in aerospace. This is because in some scenarios, the results can be catastrophic, for example, in the case of incorrect prediction of runway clear when there is an obstacle on the runway. A method which accounts for cost considerations is reject options (Cortes, DeSalvo, & Mohri, 2016).

It is important to detect when the system is operating outside of the ODD and employ strategies to mitigate such risky scenarios. Furthermore, the model must maintain high accuracy and performance within real-world scenarios, which are often different from its training conditions. Regular monitoring and updating are essential to maintain accuracy over time. Additionally, efficient use of computational resources (like CPU, memory, and storage) is critical, especially for complex models like deep learning networks.

The real-world situations might be different from the training dataset over time, and the data the model was trained on may no longer representative of the current environment (domain shift). This could occur due to a change in operation time of day, seasonality, camera lens distortions, etc. Regular monitoring for model decay and retraining with new data is necessary.

3.4.3 Aerospace standards

Certification of ML components requires showing compliance to regulatory requirements using both software and hardware certification standards as a means of compliance. Efforts in this report are focused on software aspects of ML constituents; minimal exploration of hardware certification was conducted.

3.4.3.1 RTCA DO-178: Software Considerations in Airborne Systems and Equipment Certification

DO-178C (RTCA, Inc., 2011) is the primary development assurance process by which the certification authorities such as the FAA approve all commercial software-based aerospace systems. It focuses on the safety of software used in airborne systems by establishing a set of guidelines that describe the processes and procedures that should be followed to ensure that software meets certain safety standards. The guidelines are divided into different levels of criticality, from Level A (most critical) to Level E (least critical), based on the potential impact of a software failure. The document emphasizes several key areas, including requirements management, software design, coding, integration, verification, configuration management, and quality assurance. Each of these areas has specific processes that must be followed and documented to demonstrate compliance. The primary goal of DO-178C is to ensure that the software performs its intended function with a level of confidence in safety that complies with airworthiness requirements. It also ensures that the software does not introduce any hazards that could lead to unsafe conditions.

However, there are some challenges when applying DO-178C to ML constituents, particularly neural networks:

- The neural network structures and weights cannot be traced to the system requirements from which they are developed.
- The DO-178C verification methods are not appropriate for training datasets and neural networks weights.
- The properties relevant to design assurance of datasets are fundamentally different from the properties of high-level or low-level requirements.
- Neural network weights are not comprehensible by humans, so manual review and analysis for compliance to parent requirements is not feasible.
- The structural coverage measurement metrics for source code may not be effective, because any activation of the neural network would trigger activation of all nodes to some extent. Hence, there is no value in performing a classical structural coverage approach.

Even though challenges exist in applying DO-178C completely for ML constituents, certain DO-178C objectives are still relevant:

- Software planning
- Software requirement generation
- Software configuration management
- Software quality assurance

3.4.3.2 SAE ARP 4754 Guidelines for Development of Civil Aircraft and Systems

ARP 4754A (ARP-4754 A, Guidelines for Development of Civil Aircraft and Systems, 2010) is a key guideline in the aerospace industry for developing civil aircraft and systems, especially those that are safety-critical. It outlines a systematic approach to ensure safety and compliance encompasses phases like requirement definition, design, implementation, and validation. Integral to this is the safety assessment process, which identifies and mitigates potential hazards. ARP 4754A also aligns with other standards like DO-178C for software and DO-254 for hardware, making it crucial in aircraft system certification processes. Recognized by regulatory bodies like the FAA and EASA, Challenges exist when applying ARP 4754A to systems with ML constituents.

The requirements definition approach may need some adaptations for ML In particular, the fact that some of the requirements are implicitly contained in the dataset may necessitate new

methods for evaluating the correctness and completeness of a dataset to the requirements. A specific focus on dataset requirements and AI/ML performance requirements associated with the Development Assurance Level (DAL) may also be needed.

In addition, new methods for requirements validation could be needed for ML because of the potential changes in the requirements capture philosophy. These new methods should be considered in addition to the ones defined in the ARP 4754A.

New methods for implementation verification could be needed for ML in order to cope with the probabilistic nature of ML-based algorithms and because of the potential changes in the requirements capture philosophy.

3.4.3.3 EASA Concept Paper

The EASA Concept Paper on guidance for Level 1 and 2 ML applications addresses key aspects of integrating AI and ML in aviation. It focuses on data-driven AI approaches, particularly deep learning, due to their recent significant advancements and challenges in trustworthiness. The paper emphasizes the need to adapt assurance frameworks, manage uncertainties, create data management frameworks, address model bias, and deal with the stochastic nature of AI behavior (EASA, 2023).

To create a trustworthy AI framework in aviation, the EASA AI Roadmap 1.0 (European Union Aviation Safety Agency (EASA)) identifies four essential building blocks:

- **AI Trustworthiness Analysis:** This building block serves as an interface with the EU Ethical Guidelines, encompassing ethics, safety, and security assessments as key elements for trustworthy AI development.
- **AI Assurance:** This building block addresses AI-specific guidance and includes two major topics: learning assurance, which marks a shift from programming to learning, and development/post-ops explainability, which involves providing understandable information to users about how AI/ML applications produce results. This building block also includes data recording for continuous safety monitoring and support in incident or accident investigations.
- **Human Factors for AI:** This building block introduces guidance for human factors needs associated with AI integration. It covers AI operational explainability, providing end users with understandable and relevant information about AI/ML applications' operations. It also introduces the concept of human-AI teaming to ensure effective cooperation between human users and AI systems.

- **AI Safety Risk Mitigation:** This building block acknowledges that the "AI black box" might not always be fully opened, and thus residual risks associated with the inherent uncertainty of AI need to be addressed.

The document's goal is to address the challenges of data-driven learning approaches and ensure that AI/ML solutions in aviation are trustworthy and ready for use. The paper emphasizes that all four building blocks are crucial for gaining confidence in the trustworthiness of AI/ML applications.

3.4.3.4 Overarching Properties

The Overarching Properties (OP) are a set of high-level principles designed as an alternative assurance approach for aerospace systems (Daw, Beecher, Holloway, & Graydon, 2021). They include three main attributes (Holloway, 2019):

- **Intent:** This ensures that the defined intended behavior of a system is correct and complete concerning the desired behavior. It encompasses the needs and constraints expressed by stakeholders, including safety assessments and regulatory mandates.
- **Correctness:** This property ensures that the system's implementation is correct concerning its defined intended behavior under foreseeable operating conditions. This includes both the normal and abnormal conditions anticipated during the system's operation.
- **Innocuity:** This principle states that any part of the system that is not required by the defined intended behavior should not have an unacceptable impact, particularly in compromising the safety assessment. It focuses on the systematic identification of failure conditions and their classification in an operational context.

Requisites for showing possession of OP include a defined intended behavior, failure conditions, safety assessment records, and foreseeable operating conditions. DALs are assigned using failure condition classifications.

Saswata, et al. (2023) have focused on applying the OP to AI/ML-based components in aerospace systems, specifically using the Recorder Independent Power Supply (RIPS) system as a use case. The RIPS system uses artificial neural networks to predict battery health, thereby increasing the time between maintenance actions by minimizing unnecessary battery charge/discharge cycles. Traditional certification processes like DO-178C/254 are insufficient for this case due to their inability to handle artificial neural networks. The authors used OP-based methodology for assurance of aerospace applications that include AI/ML components. However,

the proposed methodology still needs to be analyzed for practical implications of the premises and requires further investigation for generating supporting evidence.

3.4.4 Recommendations

3.4.4.1 Recommendations for defining requirements

R 1.1 - Capture ML model architecture information (architecture/structure, interface etc.) and assumptions

R 1.2 – Define requirements for the ML model for data within ODD (quality, quantity, content, collection methodology, etc.) and derived requirements for training (stopping criteria, hyperparameters selection), and system safety considerations.

Traditional software development processes are requirements-driven. Typically, there are high-level requirements and low-level requirements that are crucial for the certification process and defining the overall goals and technical details that the model must meet.

High-level requirements provide a broad view of what the model should achieve. They are more about the goals and compliance aspects of the model. They are written in a way that sets the overall objectives and boundaries for the model's development and operation. Low-level requirements delve into specifics. They define the exact parameters, pseudocode, values, and technical specifications that the model must meet. Low-level requirements are more detailed and provide clear criteria for testing and validation. Currently the challenge lies in defining low-level requirements for ML constituents, particularly neural networks; this usually also raises the question of if low-level requirements need to specify the weights of neural networks.

Central to V&V activities are the requirements defined for the system. Requirements for ML components are explored by Farrell, Mavridou, & Schumann (2023) defining the potential attributes and characteristics. This recommendation states at minimum the architectural information of the ML model should be documented. This information should be clearly defined so that the ML model architecture can be accurately implemented and maintained. In addition to the architectural information, requirements should be defined for the data within ODD. These requirements are regarding the collection of data, the data formats, the ranges of data, etc.

We recommend defining the requirements based on the input/output relation for the neural network and not the weights. An example low-level requirement is as follows: The ML constituent shall correctly identify objects during rain or fog at a minimum of 92% accuracy, as verified through real-world and simulations-based testing. As a result of defining requirements

on the input/output relation, the verification of these requirements could be conducted based on test cases of ML constituent input/output behavior.

Potential Tools:

- Verification problems require checking to ensure the input-output relations of a function hold and can be formulated as reachability, optimization (adversarial result), and search (determining counterexample) (Liu, et al., Algorithms for Verifying Deep Neural Networks, 2021).
- The Marabou framework is used for verifying deep neural networks. It can answer queries about neural network properties by transforming the queries to a satisfiability problem. Example test cases for collision avoidance neural networks using Marabou include robustness analysis and minimum separation distance (Liu, Cofer, & Osipychev, Verifying an Aircraft Collision Avoidance Neural Network with Marabou, 2023).

3.4.4.2 Recommendations for Testing

R 2.1 – Conduct testing based on functional requirements, safety requirements, model requirements (interface with system), and data requirements.

R 2.2 - Use scenario-based testing to augment current requirement-based testing. A scenario can be a model of the time sequence of scenes, centered around the system of interest, with a start time and an end time. The different types of scenarios should be derived from expert knowledge or real-world measurements.

R 2.3 - Recommend testing to be sampled from the input space based on ODD, such as statistical representation of input space. This representation could be based on identified and quantified key dimensions, such as environmental conditions (e.g. lighting, weather), object types (e.g. planes, vehicles), operational conditions (e.g. taxing, cruise).

Traditional software testing based on DO-178C is requirements-driven; it shows requirements are met and ensures that there is no hazardous unintended functionality. To meet this, DO-178C promotes the development of two kinds of test cases: normal and robustness test cases. Within these test cases, Rierson (2017) proposes several testing strategies, such as equivalence class partitioning, boundary value testing, state transition testing, decision table testing, integration testing, performance testing and complexity measurement.

Testing is part of the verification effort, which is performed to uncover errors made during development. ML component testing poses challenges, as its behavior is largely impacted by factors such as the available training data sets, the choice of hyperparameters, the model

architecture, algorithms, and optimizers. It is recommended that at minimum the testing of ML components be conducted for data testing, model testing, and integration testing (Riccio, et al., 2020). Data testing primarily consists of data requirements for the dataset, for example, the detection of unbalanced training data. Model testing would be based on model requirements and test performance of the model. Integration testing would be based on functional and safety requirements. A critical integration scenario would be. when the ML component's output serves as the primary input for the system's decision-making logic, where unforeseen OOD data could lead to unsafe control actions.

The challenge in applying these strategies for ML constituents, particularly those related to perception, is that the input space is discontinuous. Therefore, we recommended scenario-based testing to augment existing requirement-based testing. This approach is particularly useful for understanding how a model behaves under varied, often complex, circumstances that it might encounter in practical applications. These scenarios can be sampled from the ODD.

However, while using such an approach, caution should be taken to avoid coverage bias, i.e. cases where the population represented in the dataset does not match the population that the ML model is making predictions about. Also, a summary of the population or summary of data that is produced should have an accompanying notion of the user's confidence in this estimate (through measures such as confidence intervals and p-values).

3.4.4.3 Recommendations for assessing system malfunctions and prediction error

R 3.1 – Attribute a malfunction of an ML-enabled system to various failures of hardware, sensors, requirement errors, etc. Analyze these instances by traditional safety processes through familiar tools such as FHA, FMEA, etc.

R 3.2 – Determine the prediction error of the ML model, in part through assessments of outputs compared to expected functions and performance as well as an assessment of calibrated uncertainty against predicted values.

Despite typically having small codebases, ML constituents—particularly neural networks—present significant challenges in identifying and understanding latent errors. Unlike traditional software, even when these errors are identified, retraining the model does not guarantee their elimination, as the underlying logic remains opaque. Consequently, detecting and managing ML malfunctions is a critical requirement for maintaining system safety

When performing a safety assessment of an ML-enabled system, it is often forgotten that an ML item is a component in the system pipeline. This recommendation just re-emphasizes that the

system consists of various other hardware and software elements. A system malfunction could be attributed to other components in the system and not just the ML item. For example, in a perception system using an ML-based object detector, a camera experiencing sensor fault could provide corrupted image as an input to the ML model. However, safety measures should be in place to detect such corruption in input for ML model. This could be achieved through UQ.

We recommend that systems and safety engineers be involved in reviewing safety assessments to determine how the ML constituent may malfunction. This review implies expanding the safety assessment to include the software in order to help understand the impact of the malfunction and develop steps to mitigate risk. The ML constituent developers should be aware of the basic system development.

Beyond incorporating the safety assessment information into ML constituent development, an approach to determine if the ML constituents are malfunctioning is to use their estimate of uncertainty in prediction. This UQ of the estimate score from the ML constituent can provide confidence information about its prediction. If the prediction is of low confidence, it could be attributed to an ML constituent malfunction, and a potential cause could be that the input data is OOD.

3.4.4.3.1 Potential techniques for UQ

Some of the potential techniques that can be used for uncertainty estimates include Bayesian neural networks, ensemble methods (Monte-Carlo Dropout, deep ensemble, hyperparameter ensemble), and test-time augmentation methods (Gawlikowski, et al., 2023).

- Bayesian neural networks offer a probabilistic interpretation of neural networks. They provide a distribution over weights rather than fixed weight values, allowing for an estimation of uncertainty.
- Ensemble methods, such as random forests or an ensemble of deep learning models, quantify uncertainty by looking at the variance in predictions across the ensemble. The disagreement between different models in the ensemble reflects uncertainty.
- Test-time data augmentation involves applying various transformations to test data and observing the variance in predictions. The variation in predictions for different augmented versions of the same data point can provide an estimate of uncertainty.

3.4.4.4 Recommendations for assurance

R 4.1 - Demonstrate assurance for ML models based on the following artifacts:

- Data management – Coverage of operational design domain, including corner/edge cases for training and testing
- Robustness – Normal and adverse conditions within the ODD, including credible edge and corner cases
- Performance –Element behavior as specified and to the degree that it is specified (e.g., fast and accurate)
- Stability – Consistency of behavior in presence of small input perturbations
- Resilience of safety-critical functions – Behavior in the presence of faults

R 4.2 - Use RTA to provide safety assurance of ML models during runtime in addition to tailored development assurance. The requirement for RTA should be determined based on software criticality level and system design.

R 4.3 - Include additional considerations during runtime so as to include execution of decisions and actions and risk mitigation

ML models in safety-critical systems present significant assurance challenges due to their inherent complexity and probabilistic nature. Traditional V&V methods often fall short in ensuring reliability and safety in these models. Therefore, we recommend the assurance for ML constituents be demonstrated based on the following objectives: data management, robustness of the ML constituent (expanding to system, sub-function, trained model, and inference model), performance as defined in the previous section, repeatability of behavior, and resilience to the presence of faults. An example case study for demonstrating robustness for neural networks used for collision avoidance was presented in Section 3.1.5.

Beyond the evidence produced for assurance objectives defined above, we also recommend using RTA architectures to provide safety assurance during runtime. However, the need for RTA must be determined based on the safety assessments conducted. An example implementation of safety assurance during runtime using RTA for a collision avoidance neural networks can be found in *Flight Test of a Collision Avoidance Neural with Run-Time Assurance* (Cofer, et al., 2022).

Additional considerations during runtime could include monitoring the deployed model to ensure that it continues to perform as intended. Monitors could include *input monitoring*, for checking whether inputs are within acceptable bounds before they are given to the ML model;

environment monitoring, for checking that the observed environment matches the ODD, or *output monitoring*, checking the uncertainty information available.

3.4.4.5 Recommendations for stakeholder competence

R 5.1 – Develop and deploy formal training in ML for all the stakeholders in the V&V process, specifically on safety violation and errors in ML systems.

R 5.2 - Determine which ODA discipline will be delegated to find compliance for AI-based systems. Expansion of existing DER/E-UM software or new AI DER/E-UM should be considered. Once a determination is made, update FAA Order 8100.8D Table 4.1 (FAA, 2011) to document specific applicant requirements for a Designated Engineering Representative (DER) candidate seeking authorization for approval of AI-based systems.

R 5.3 – Provide the training necessary to understand the AI-based system life cycle and certification process to FAA personnel tasked with oversight of the certification of AI-based systems. Propose that the training be provided to staff in roles such as those involved with the publication of guidance, DER/E-UM advisors, Continued Operational Safety, and Certification Branches (formally known as ACOs).

R 5.4 - Commission a study on existing development assurance process gaps with the industry for specific artifacts pertaining to DO-178 (RTCA, Inc., 2011) and DO-254 (RTCA, Inc., 2000), as mentioned in the FAA AI Roadmap (FAA, 2024).

R 5.5 - If possible, allow industry members to attend the FAA's ML training session. This would help establish a common understanding of ML among all stakeholders involved.

R 5.6 - Provide information on the innovation intake process.

The inclusion of AI into airborne systems will be a significant departure from traditional systems. As such, it will also require a significant departure in the way that engineers who integrate the technology into their systems, and the regulators who provide oversight to those systems, need to be trained in order to provide them. R 5.1-5.6 seek to provide suggestions on how this departure may be addressed. Recommendations R 5.1 and R 5.3 identify two main stakeholder groups that should receive additional training in order to allow them to perform their roles effectively: the portion of the applicant organization that is involved with the V&V process, and the portion of the regulator that will be tasked with issuing guidance or overseeing the certification of these systems. This requirement does not imply a totally comprehensive training, as it is understood that AI is a rapidly evolving field and this is impossible. Nor is it intended to suggest the requirement for a university degree in AI for all stakeholders. However, the

fundamentals of AI are well established and readily communicable, and the intent of this recommendation is that those fundamentals are what are provided to these stakeholder groups. Providing this training through an organization-sponsored training class will help both stakeholder groups be better prepared to ensure the correct requirements are being levied on the systems.

Due to the relative novelty and lack of precedent in the certification of airborne AI systems, it is assumed that the FAA will choose to retain all of the early certification projects involving AI. However, as certification of these systems moves from novel to routine, it is also assumed that the FAA will want to begin delegating compliance assessment to ODAs with the proper experience and qualifications. AI is a novel technology that is unlike other technologies currently in use. This difference prompts the question: what is the proper experience and qualifications for a delegated finder of compliance for these systems?

FAA Order 8000.95C - Designee Management Policy (FAA, 2023) denotes the additional technical criteria for DER appointment for various subsystems. On first reading, none of the additional requirements listed for any DER subsystem would seem to fully address the complexities created by the inclusion of AI. Recommendation R 5.2 suggests the FAA to consider this question and provide a determination as to whether the assessment of compliance for AI systems will fall under the current authorization of an existing DER/E-UM subsystem, such as Software, or if it will require the expansion of an existing or creation of a new DER/E-UM subsystem. Assuming that, at a minimum, additional training or experience requirements are created as an expansion to an existing DER/E-UM subsystem, candidates will need to gather the work experience necessary to meet those requirements. In order for the FAA to be able to delegate future compliance findings, there must first be approved designees available to be delegated to, and that will not be possible without first determining and publishing what the qualifications will be.

3.4.5 Challenges

Given the recommendations, there still exist the following challenges in implementing them:

- Defining and validating of requirements for ML constituents is a challenge, as there is complexity in understanding data characteristics, the dynamic nature of requirements, and lack of existing standardized practices.
- Defining ODD is difficult due to a diversity of environments, the unpredictable nature of real-world situations, and data limitations.

- Open-loop verification of very large neural networks is challenging, as the cost of branch-and-bound-based algorithms scales exponentially with the required split depth, making branching decisions both harder and more important (Brix, Müller, & Bak, 2023).
- Verification outside of the classification setting is underexplored, leading to a lack of established approaches for image segmentation, object detection, etc. (Brix, Müller, & Bak, 2023).
- Closed-loop verification for state-space explosion and scalability issues needs to be addressed in neural network controllers (Lopez, et al., 2023).

4 Conclusion

The integration of AI and ML technologies into safety-critical airborne systems represents a significant step forward in advancing autonomy, operational efficiency, and safety in aviation. However, this integration also introduces unique challenges, particularly regarding verification, validation, certification, and risk management. This report highlights key findings and provides some recommendations to address these challenges.

A robust safety assurance framework is essential to ensure that AI/ML systems meet the rigorous demands of aviation safety. Traditional assurance methods must evolve to account for the complexities of data-driven algorithms, emphasizing properties like explainability, UQ, and robustness. Incorporating new tools and techniques, such as RTA, CM, and advanced V&V methods, will enhance the reliability of AI/ML systems and provide confidence in their performance under dynamic conditions.

Human-AI interaction is another critical area that demands careful consideration. Trust, transparency, and clear roles in human-AI teaming must be established to foster effective collaboration in safety-critical operations. Addressing these human factors is essential for the successful adoption of AI/ML technologies in aviation.

The path forward includes continued research into advanced assurance methodologies, close collaboration between industry and regulatory agencies, and the establishment of AI-specific certification standards. As AI/ML technologies mature, ongoing innovation and iterative refinement of safety assurance frameworks will be necessary to support their safe and effective deployment.

In conclusion, while the transformative potential of AI/ML in aviation is clear, achieving full integration requires a multifaceted approach encompassing technological advancements, regulatory evolution, and human-centric design. By addressing the identified challenges and leveraging the proposed solutions, the aviation industry can safely harness the benefits of AI/ML, driving innovation while maintaining the highest standards of safety.

5 References

Urban Challenge. (2007, November 3). Retrieved from Defense Advanced Research Projects Agency: https://www.darpa.mil/about-us/timeline/darpa-urban-challenge

A. Carrio, C. S.-R. (2017). A Review of Deep Learning Methods and Applications for Unmanned Aerial Vehicles. *Hindawi Journal of Sensors*.

A. Krizhevsky, I. S. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems (NIPS).*

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., . . . Jia, Y. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems.

Abrecht, S., Hirsch, A., Raafatnia, S., & Woehrle, M. (2023). *Deep Learning Safety Concerns in Automated Driving Perception*. Retrieved from https://arxiv.org/pdf/2309.03774.pdf

Ackerman, S., Raz, O., Zalmanovici, M., & Zlotnick, A. (2021). *Automatically detecting data drift in machine learning classifiers.* arXiv:2111.05672.

Adedjouma, M., Botella, B., Ibanez-Guzman, J., Mantissa, K., Proum, C. M., & Smaoui, A. (2024). Defining Operational Design Domain for Autonomous Systems: A Domain-Agnostic and Risk-Based Approach. *19th Annual System of Systems Engineering Conference.* Tacoma, WA.

(2020). *AFE 87 - Machine Learning.* AVSI.

Agogino, A., Brat, G., He, Y., Hulse, D., Lipkis, R., Pressburger, T., & Gopinath, D. (2024). *NASA/TM–20240006865: Recommendations on Evidence and Process for Certification of Learning-enabled Components in Aerospace Systems.* NASA.

Aleksandrova, M., & Chertov, O. (2021). Impact of model-agnostic nonconformity functions on efficiency of conformal classifiers: an extensive study. *Proceedings of the Tenth Symposium on Conformal and Probabilistic Prediction and Applications.*

Alsayed, Z., Resende, P., & Bradai, B. (2021). Operational Design Domain Monitoring at Runtime for 2D Laser-Based Localization algorithms. *20th International Conference on Advanced Robotics.* Ljubljana, Slovenia.

Altun, A. T., Xu, Y., Inalhan, G., Vidal-Franco, I., & Hardt, M. (2022). Contingency Management Concept Generation for U-Space System. *Integrated Communication, Navigation and Surveillance Conference (ICNS).* Dulles, VA.

Alvarez, L. E., Jessen, I., Owen, M. P., Silbermann, J., & Wood, P. (2019). ACAS sXu: Robust Decentralized Detect and Avoid for Small Unmanned Aircraft Systems. *IEEE/AIAA 38th Digital Avionics Systems Conference (DASC).* San Diego.

American Psychological Association. (2009). *Publication manual of the American Psychological Association.* Washington, D.C.: American Psychological Association.

Amir, G., Schapira, M., & Katz, G. (2021). Towards Scalable Verification of Deep Reinforcement Learning. *Formal Methods in Computer-Aided Design.*

Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., & Mané, D. (2016). Concrete Problems in AI Safety. *arXiv:1606.06565v2.*

Angelopoulos, A. N., & Bates, S. (2022). A Gentle Introduction to Conformal Prediction and Distribution-Free Uncertainty Quantification. *arXiv:2107.07511.*

Angelopoulos, A., Bates, S., Malik, J., & Jordan, M. I. (2020). *Uncertainty Sets for Image Classifiers using Conformal Prediction.* arXiv:2009.14193.

ANSI UASSC Working Group. (2020). *Standardization Roadmap for Unmanned Aircraft Systems, Version 2.0,.* ANSI.

Antonin Raffin and Ashley Hill ,Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, Noah Dormann. (2021). Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research.*

Arnez, F., Espinoza, H., Radermacher, A., & Terrier, F. (2020). A Comparison of Uncertainty Estimation Approaches in Deep Learning Components for Autonomous Vehicle Applications. *CEUR Artificial Intelligence Safety.* Yokohama, Japan.

(2010). *ARP-4754 A, Guidelines for Development of Civil Aircraft and Systems.*

(2023). *ARP-4754 B, Guidelines for Development of Civil Aircraft and Systems.* SAE.

ASTM. (2021). *ASTM F3269-21: Standard Practice for Methods to Safely Bound Behavior of Aircraft Systems Containing Complex Functions Using Run-Time Assurance.* ASTM.

ASTM. (2022). *F3269 − 21: Standard Practice for Methods to Safely Bound Behavior of Aircraft Systems Containing Complex Functions Using Run-Time Assurance.* ASTM.

Aviation Rulemaking Advisory Committee (ARAC). (2002). Draft Advisory Circular AC 25.1309-1B: System Design and Analysis (Arsenal Version). Federal Aviation Administration.

Bacci, E., & Parker, D. (2020). Probabilistic Guarantees for Safe Deep Reinforcement Learning. *arXiv:2005.07073v2.*

Bacci, E., & Parker, D. (2022). Verified Probabilistic Policies for Deep Reinforcement Learning. *NASA Formal Methods: 14th International Symposium,.* Pasadena, CA.

Bach, T. A., Kristiansen, J. K., Babic, A., & Jacovi, A. (2024). Unpacking Human-AI Interaction in Safety-Critical Industries: A Systematic Literature Review. *IEEE Access, 12*, 106385-106414.

Baheri, A., Nageshrao, S., Tseng, H. E., Kolmanovsky, I., Girard, A., & Filev, D. (2020). Deep Reinforcement Learning with Enhanced Safety for Autonomous Highway Driving. *IEEE Intelligent Vehicles Symposium.* Las Vegas.

Balduzzi, G., Bravo, M. F., Chernova, A., Cruceru, C., van Dijk, L., de Lange, P., . . . Yakushev, G. (2021). *Neural Network Based Runway Landing Guidance for General Aviation Autoland.* New Jersey: FAA.

Balduzzi, G., Ferrari, M. B., Chernova, A., Cruceru, C., van Dijk, L., de Lange, P., . . . PIllio, Z. (2021). *DOT/FAA/TC-21/48 Neural network based runway landing guidance for general aviation autoland.* New Jersey: FAA,William J. Hughes Technical Center.

Ballard, Z., Brown, C., Madni, A. M., & Ozcan, A. (2021). Machine learning and computation-enabled intelligent sensor design. *Nature Machine Intelligence, 3*, 556-565.

Barandas, M., Folgado, D., Santos, R., Simão, R., & Gamboa, H. (2022). Uncertainty-Based Rejection in Machine Learning: Implications for Model Development and Interpretability. *Electronics, 11*(3).

Bastani, O., Pu, Y., & Solar-Lezama, A. (2018). Verifiable Reinforcement Learning via Policy Extraction. *32nd Conference on Neural Information Processing Systems.* Montreal, Canada.

Blei, D. M., Kucukelbir, A., & McAuliffe, J. D. (2017). Variational Inference: A Review for Statisticians. *Journal of the American Statistical Association, 112*, 859-877.

Bochkovskii, A. (n.d.). *AlexeyAB/darknet: YOLOv4 / Scaled-YOLOv4 / YOLO - Neural Networks for Object Detection (Windows and Linux version of Darknet)*. Retrieved from GitHub: https://github.com/AlexeyAB/darknet

Bochkovskiy, A., Wang, C.-Y., & Hong-Yuan, M. L. (2020, April 23). *YOLOv4: Optimal Speed and Accuracy of Object Detection.* Retrieved from https://arxiv.org/abs/2004.10934

Bochovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). *YOLOv4: Optimal Speed and Accuracy of Object Detection*. Retrieved from https://doi.org/10.48550/arXiv.2004.10934

Boniol, F., Chan-Hon-Tang, A., Eudes, A., Herbin, S., Le Besnerais, G., Pagetti, C., & Sanfourche, M. (2020). Challenges in the Certification of Computer Vision-Based Systems. *Aerospace Lab*.

Brat, G. (2019). *NASA and Boeing to Work Together on Verification and Validation Technology for Autonomy*. Retrieved from https://ti.arc.nasa.gov/news/NASA-Boeing-SAA/

Braun, D., Marb, M. B., Angelov, J., Wechner, M., & Holzapfel, F. (2023). Worst-Case Analysis of Complex Nonlinear Flight Control Designs Using Deep Q-Learning. *Journal of Guidance, Control and Dynamics, Vol 46.*

Breck, E., Polyzotis, N., Roy, S., Whang, S., & Zinkevich, M. (2019). Data Validation for Machine Learning. *MLSys.*

Brix, C., Müller, M. N., & Bak, S. (2023). First three years of the international verification of neural networks competition (VNN-COMP). *International Journal on Software Tools for Technology Transfer, Vol 25.*

Brunke, L., Greeff, M., Hall, A. W., Yuan, Z., Zhou, S., Panerati, J., & Schoellig, A. P. (2022). Safe Learning in Robotics: From Learning-Based Control to Safe Reinforcement Learning. *Annual Review of Control, Robotics, and Autonomous Systems.*

C. Wilkinson, J. Lynch, R. Bharadwaj and K. Woodham. (2016). *Verification of Adaptive Systems.* FAA.

Campbell, N. H., Acheson, M. J., & Gregory, I. M. (2021). Dynamic Vehicle Assessment for Intelligent Contingency Management of Urban Air Mobility Vehicles. *AIAA SCITECH.* VIRTUAL EVENT.

Cappi, C., Chapdelaine, C., Gardes, L., Jenn, E., Lefevre, B., Picard, S., & Soumarmon, T. (2021). *Dataset definition standard.* DEEL Certification Workgroup.

Cappi, C., Cohen, N., Ducoffe, M., Gabreau, C., Gardes, L., Gauffriau, A., . . . Vigouroux, D. (2024). How to design a dataset compliant with an ML-based system ODD? *12th European Congress on Embedded Real Time Software and Systems.* Toulouse, France.

Cappi, C., Cohen, N., Ducoffe, M., Gabreau, C., Gardes, L., Gauffriau, A., Ginestet, J.B., Mamalet, F., Mussot, V., Pagetti, C., Vigouroux, D. (2024). How to design a dataset compliant with an ML-based system ODD? *12th European Congress on Embedded Real Time Software and Systems.* Toulouse, France.

Carranza-García, M., Torres-Mateo, J., Lara-Benítez, P., & García-Gutiérrez, J. (2021). On the Performance of One-Stage and Two-Stage Object Detectors in Autonomous Vehicles Using Camera Data. *Remote Sensing, 13*(https://doi.org/10.3390/rs13010089).

Carter, H. G., Chan, A., Vinegar, C., & Rupert, J. (2023). Proposing the Use of Hazard Analysis for Machine Learning Data Sets. *Journal of System Safety, 58*(2).

Chai, R., Niu, H., Carrasco, J., Arvin, F., Yin, H., & Lennox, B. (2022). Design and Experimental Validation of Deep Reinforcement Learning-Based Fast Trajectory Planning and Control for Mobile Robot in Unknown Environment. *IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS.*

Chandrasekaran, R., Payan,, A., Collins,, K., & Mavris, D. (2019). *A Survey of Wire Strike Prevention and Protection Technologies for Helicopters.* Technical Report, U.S. Department of Transportation, Federal Aviation Administration. Retrieved from http://actlibrary.tc.faa.gov

Chen, S., Sun, Y., Li, D., Wang, Q., Hao, Q., & Sifakis, J. (2022). Runtime Safety Assurance for Learning-enabled Control of Autonomous Driving Vehicles. *IEEE International Conference on Robotics and Automation.* Philadelphia.

Cheng, R., Orosz, G., Murray, R. M., & Burdick, J. W. (2019). End-to-End Safe Reinforcement Learning through Barrier Functions for Safety-Critical Continuous Control Tasks. *Thirty-Third AAAI Conference on Artificial Intelligence.*

Choi, J., Castaneda, F., Tomlin, C. J., & Sreenath, K. (2020). Reinforcement Learning for Safety-Critical Control under Model Uncertainty, using Control Lyapunov Functions and Control Barrier Functions. *Robotics: Science and Systems.* Corvalis, Oregan.

Chuan Guo, Geoff Pleiss, Yu Sun, Kilian Q. Weinberger. (2017). On Calibration of Modern Neural Networks. *Proceedings of the 34th International Conference on Machine Learning.* Sydney, Australia.

Chung, Y., Char, I., Guo, H., Schneider, J., & Neiswanger, W. (2021). Uncertainty Toolbox: an Open-Source Library for Assessing, Visualizing, and Improving Uncertainty Quantification. *arXiv:2109.10254.*

Ciampiconi, L., Elwood, A., Leonardi, M., Mohamed, A., & Rozza, A. (2023). A survey and taxonomy of loss functions in machine learning. *arXiv:2301.05579v1.*

Clayton, A. (2021). *Bernoulli's Fallacy - Statistical Illogic and the Crisis of Modern Science.* Columbia University Press.

Clement, D., Mottino, S., & Costello, D. H. (2024). Development Life Cycle for Using a DNN to Complete the Autonomous Aerial Refueling Task. *IEEE Access, 12*, 99163-99174.

cocodataset. (n.d.). *COCO API - Dataset @ http://cocodataset.org/*. Retrieved from GitHub: https://github.com/cocodataset/cocoapi

Cofer, D., Amundson, I., Sattigeri, R., Passi, A., Boggs, C., Smith, E., . . . Rayadurgam, S. (2020). Run-Time Assurance for Learning-Enabled Systems. *12th International Symposium, NFM.* Moffett Field,CA.

Cofer, D., Sattigeri, R., Amundson, I., Babar, J., Hasan, S., Smith, E. W., . . . Timmerman, L. (2022). Flight Test of a Collision Avoidance Neural with Run-Time Assurance. *IEEE/AIAA 41st Digital Avionics Systems Conference (DASC).*

Computer Vision Center and The Embodied AI Foundation. (2021, April). *CARLA Simulator*. Retrieved from https://carla.org/

Cone, C., Owen, M., Alvarez, L., & Brittain, M. (2023). Reward Function Optimization of a Deep Reinforcement Learning Collision Avoidance System. *AIAA SCITECH.* National Harbor, Maryland.

Cornelia Gruber, Patrick Oliver Schenk, Malte Schierholz, Frauke Kreuter, Göran Kauermann. (2023). *Sources of Uncertainty in Machine Learning - A Statisticians' View.* arXiv:2305.16703.

Corsi, D., Marchesini, E., & Farinelli, A. (2021). Formal Verification of Neural Networks for Safety-Critical Tasks in Deep Reinforcement Learning. *Thirty-Seventh Conference on Uncertainty in Artificial Intelligence.*

Cortes, C., DeSalvo, G., & Mohri, M. (2016). Learning with Rejection. *Algorithmic Learning Theory: 27th International Conference.*

D. Gopinath, H. C. (2019). Property inference for deep neural networks. *International Conference on Automated Software Engineering.*

D. Silver, A. H. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 484-489.

Dai, Y., Ouyang, H., Zheng, H., Long, H., & Duan, X. (2022). Interpreting a deep reinforcement learning model with conceptual embedding and performance analysis. *Applied Intelligence.*

Dame, S. D. (2020, December 29). *United States Patent No. US10878709B2.*

Danesh, M. H., & Fern, A. (2021). Out-of-Distribution Dynamics Detection: RL-Relevant Benchmarks and Results. *ICML.*

DARPA. (2019). *Assured Autonomy*. Retrieved from https://www.darpa.mil/program/assured-autonomy

Davoodi, P., Lai, G., Morris, T., & Sharma, S. (2019, June 3). *High performance inference with TensorRT Integration*. Retrieved 2022, from https://medium.com/tensorflow/high-performance-inference-with-tensorrt-integration-c4d78795fbfe

Daw, Z., Beecher, S., Holloway, M., & Graydon, M. (2021). Overarching Properties as means of compliance: An industrial case study. *IEEE/AIAA 40th Digital Avionics Systems Conference (DASC).* San Antonio, TX.

DIN Media. (2024). *Artificial Intelligence - Uncertainty quantification in machine learning.* DIN.

Ding, Y., Liu, J., Jinjun, X., & Shi, Y. (2019). *Revisiting the Evaluation of Uncertainty Estimation and Its Application to Explore Model Complexity-Uncertainty Trade-Off.* arXiv:1903.02050.

Dolph, C., Glaab, L., Allen, B., Consiglio, M., & Iftekharuddin, K. (2019). An Improved Far-Field Small Unmanned Aerial System Optical Detection Algorithm. *IEEE/AIAA 38th Digital Avionics Systems Conference.* San Diego, CA.

Dong, Y., Zhao, X., Wang, S., & Huang, X. (2022). Reachability Verification Based Reliability Assessment for Deep Reinforcement Learning Controlled Robotics and Autonomous Systems. *CoRR.*

Dulac-Arnold, G., Levine, N., & Mankowitz, D. J. (2021). Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning Vol 110.*

Dunlap, K., Mote, M., Delsing, K., & Hobbs, K. L. (2022). Run Time Assured Reinforcement Learning for Safe Satellite Docking,. *AIAA SCITECH.* San Diego.

EASA. (2023). *EASA Concept Paper: First Usuable guidance for Level 1&2 machine learning applications.*

EASA. (2024). *EASA Concept Paper: Guidance for for Level 1 & 2 Machine Learning Applications, Issue 2.* EASA.

EASA AI Task Force and Daedalean AG. (2020). *Concepts of Design Assurance for Neural Networks (CoDANN),.* EASA.

Ensemble-PyTorch Developers. (2021). *Ensemble PyTorch*. Retrieved 2024, from https://ensemble-pytorch.readthedocs.io/en/latest/introduction.html

Erignac, C. A. (2024, February 20). *United States Patent No. 11,908,336.*

European Union Aviation Safety Agency (EASA). (n.d.). EASA Artificial Intelligence Roadmap: A deliverable of the EASA AI Task Force (Version 1.0). Retrieved from https://www.easa.europa.eu/en/document-library/general-publications/easa-artificial-intelligence-roadmap-10

Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Tramer, F., . . . Song, D. (2018). Physical Adversarial Examples for Object Detectors. *12th USENIX Workshop on Offensive Technologies.* Baltimore.

F38.01, A. S. (2017). *ASTM F3269-17: Standard Practice for Methods to Safely Bound Flight Behavior of Unmanned Aircraft Systems Containing Complex Functions.* ASTM.

FAA. (2011, October 28). *Order 8100.8D - Designee Management Handbook*. (FAA) Retrieved December 2023, from

https://www.faa.gov/regulations_policies/orders_notices/index.cfm/go/document.information/documentid/1019601

FAA. (2023, September 21). Designee Management Policy. Retrieved from https://www.faa.gov/documentLibrary/media/Order/FAA_Order_8000.95C.pdf

FAA. (2024). Roadmap for Artificial Intelligence Safety Assurance (Version 1.0).

Fang, K., Tao, Q., Huang, X., & Yang, J. (2024). Revisiting Deep Ensemble for Out-of-Distribution Detection: A Loss Landscape Perspective. *International Journal of Computer Vision*.

Farrell, M., Mavridou, A., & Schumann, J. (2023). Exploring Requirements for Software that Learns: A Research Preview. *Requirements Engineering: Foundation for Software Quality. REFSQ.*

Federal Aviation Administration. (2011, October 28). Order 8100.8D: Designee Management Handbook. Retrieved from https://www.faa.gov/regulations_policies/orders_notices/index.cfm/go/document.information/documentid/1019183

Feng, S., Sun, H., & Yan, X. (2023). Dense reinforcement learning for safety validation of autonomous vehicles. *Nature.*

Feurer, M., & Hutter, F. (2019). Hyperparameter optimization. *Automated machine learning: Methods, systems, challenges.*

Franke, J., Garrett, R. C., Pasareanu, C., Malik, W., & Hemann, G. (2015). *SafeTug Semi-Autonomous Aircraft Towing Vehicles.* NASA NARI.

Fulton, N., & Platzer, A. (2018). Reinforcement Learning via Formal Methods: Toward Safe Control Through Proof and Learning. *Proceedings of the AAAI Conference on Artificial Intelligence.*

Ganaie, M. A., Hu, M., Malik, A. K., Tanveer, M., & Suganthan, P. N. (2022). Ensemble deep learning: A review. *Engineering Applications of Artificial Intelligence, 115*.

Gawlikoski, J., Tassi, C. R., Ali, M., Lee, J., Humt, M., Feng, J., . . . Zhu, X. X. (2023). A survey of uncertainty in deep neural networks. *Artificial Intelligence Review, 56*(1), 1513-1589.

Gawlikowski, J., Tassi, C. R., Ali, M., Lee, J., Humt, M., Feng, J., . . . Zhu, X. X. (2023). A survey of uncertainty in deep neural networks. *Artificial Intelligence Review, vol 56.*

Gawlikowski, J., Tassi, C.R.N., Ali, M. et al. (2023). A survey of uncertainty in deep neural networks. *Artificial Intelligence Review, 56*(1), 1513-1589.

Genin, D., Papusha, I., Brulé, J., Young, T., Mullins, G., Kouskoulas, Y., . . . Schmidt, A. (2021). Formal Verification of Neural Network Controllers for Collision-Free Flight. *Software Verification. NSV VSTTE.*

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Ch 6. Deep Forward Networks. In *Deep Learning.* MIT Press.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning, Chapter 7. *MIT Press.*

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. (2016). Deep learning, Chapter 11. *MIT Press.*

Gregory, I. M., Neogi, N. A., Grauer, J. A., Campbell, N. H., Holbrook, J., Bacon, B. J., . . . Cook, J. (2021). *Intelligent Contingency Management for Urban Air Mobility.* VIRTUAL EVENT: AIAA SCITECH.

Gruber, C., Schenk, P. O., Shierholz, M., Kreuter, F., & Kauermann, G. (2023). *Sources of Uncertainty in Machine Learning - A Statisticians' View.* arXiv:2305.16703.

Gu, S. (n.d.). *Safe-Reinforcement-Learning-Baselines*. Retrieved September 2023, from https://github.com/chauncygu/Safe-Reinforcement-Learning-Baselines

Guo, C., Pleiss, G., Sun, Y., & Winberger, K. Q. (2017). On Calibration of Modern Neural Networks. *Proceedings of the 34th International Conference on Machine Learning.* Sydney, Australia.

Guo, W., Zhou, Y., & Wei, P. (2022). Exploring online and offline explainability in deep reinforcement learning for aircraft separation assurance. *Frontiers in Aerospace Engineering.*

Hagan, M. T., Demuth, H. B., Beale, M. H., & De Jesus, O. (2014). *Neural Network Design* (2nd ed.).

Han Xiao, Kashif Rasul, Roland Vollgraf. (2017). *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms.* arXiv:1708.07747.

Harris, A., & Schaub, H. (2020). Spacecraft Command and Control with Safety Guarantees using Shielded Deep Reinforcement Learning. *AIAA SCITECH Forum.* Orlando, Florida.

Hasanbeig, M., Abate, A., & Kroening, D. (2020). Cautious Reinforcement Learning with Logical Constraints. *International Foundation for Autonomous Agents and Multiagent Systems.*

Haskins, A. A. (2021). *Capturing Intentional Testing of an Automated System. [Doctoral dissertation].* Wright State University.

He, Y., & Schumann, J. (2023). Statistical Analysis and Runtime Monitoring for an AI-based Autonomous Centerline Tracking System. *Proceedings of the Asia Pacific Conference of the PHM Society.*

He, Y., Schumann, J., & Yu, H. (2022). Toward Runtime Assurance of Complex Systems with AI Components. *Proceedings of the 7th European Conference of the Prognostics and Health Management Society.*

Hobbs, K. L., Heiner, B. K., Busse, L., Dunlap, K., Rowanhill, J., Hocking, A. B., & Zutshi, A. (2023). Systems Theoretic Process Analysis of a Run Time Assured Neural Network Control Systems. *AIAA SCITECH.* National Harbor, MD.

Hobbs, K., Mote, M., Abate, M., Coogan, S., & Feron, E. (2022). *Run Time Assurance for Safety-Critical Systems: An Introduction to Safety Filtering Approaches for Complex Control Systems*. Retrieved from https://doi.org/10.48550/arXiv.2110.03506

Hoff, K. A., & Bashir, M. (2015). Trust in Automation: Integrating Empirical Evidence on Factors That Influence Trust. *Human Factors, 57*(3), 407-434.

Holloway, C. M. (2019). Understanding the overarching properties. *NTRS NASA.*

Hongseok, C. (2020). *Operational Design Domain (ODD) Framework for Driver-Automation Integrated Systems.* MIT.

Hsu, K.-C., Rubies-Royo, V., Tomlin, C., & Fisac, J. F. (2021). Safety and Liveness Guarantees through Reach-Avoid Reinforcement Learning. *Proceedings of Robotics: Science and Systems.*

Hüllermeier, E., & Waegeman, W. (2021). Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods. *Machine Learning, 110*, 457-506.

Hüllermeier, E., & Waegeman, W. (2021). Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods. *Machine Learning, 110*, 457-506.

IEEE. (2023). IEEE 3219 - Standard for Robustness Testing and Evaluation of Aritificial Intelligence (AI)-based Image Recognition Service. *Artificial Intelligence Standards Committee of IEEE Computer Society.*

Implementations of Uncertainty Quantification Methods for Neural Networks. (2021). Retrieved 2024, from https://github.com/JakobCode/UncertaintyInNeuralNetworks_Resources

Introvigne, M., Ramazzina, A., Walz, S., Scheuble, D., & Bijelic, M. (2024). Real-time Environment Condition Classification for Autonomous Vehicles. *IEEE Intelligent Vehicles Symposium (IV).* Jeju Island, Korea, Republic of.

ISO. (2021). ISO/IEC TR 24029 Artificial Intelligence (AI) — Assessment of the robustness of neural networks Part 1& 2. *ISO/IEC.*

ISO. (2022). Artificial Intelligence - Assessment of machine learning classification performance ISO/IEC TS-4213. *ISO/IEC Technical Specification.*

ISO 34530. (2023). *Road Vehicles — Test scenarios for automated driving systems - Specifications for operational design domain.* International Organization for Standardization.

Iyer, R., Li, Y., Li, H., Lewis, M., Sundar, R., & Sycara, K. (2018). Transparency and Explanation in Deep Reinforcement Learning Neural Networks. *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society.*

J. Redmon, S. D. (2016). You Only Look Once: Unified, Real-Time Object Detection. *Conference on Computer Vision and Pattern Recognition (CVPR).* IEEE.

J. Schumann, P. M. (2015). R2U2: monitoring and diagnosis of security threats for unmanned aerial systems. *Runtime Verification.*

J. Zhang, M. H. (2020). Machine Learning Testing: Survey, Landscapes and Horizons. *IEEE Transactions on Software Engineering*.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., . . . Kalenichenko, D. (2018). Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* Salt Lake City.

Jaoude, A. A. (2017). The paradigm of complex probability and Claude Shannon's information theory. *Systems Science & Control Engineering, 5*(1), 380-425. doi:10.1080/21642583.2017.1367970

Japkowicz, N., & Shah, M. (2011). Evaluating learning algorithms: a classification perspective. *Cambridge University Press.*

Javier Garcia, Fernando Fernandez. (2015). A Comprehensive Survey on Safe Reinforcement Learning. *Journal of Machine Learning Research, Vol 16.*

Jaynes, E. T. (2003). *Probability Theory – the Logic of Science.* Cambridge University Press.

Jean-Paul, S., Elseify, T., Obeid, I., & Picone, J. (2019). Issues in the reproducibility of deep learning results. *2019 IEEE Signal Processing in Medicine and Biology Symposium (SPMB)*, 1-4.

Jiang, J., Karran, A. J., Coursaris, C. K., Léger, P. M., & Beringer, J. (2022). A Situation Awareness Perspective on Human-AI Interaction: Tensions and Opportunities. *International Journal of Human–Computer Interaction, 39*(9), 1789-1806.

Jiang, W., Zhao, Y., Wu, Y., & Zuo, H. (2022). Capturing Model Uncertainty with Data Augmentation in Deep Learning. *Proceedings of the SIAM International Conference on Data Mining (SDM).*

Jin, M., & Lavaei, J. (2020). Stability-Certified Reinforcement Learning: A Control-Theoretic Perspective. *IEEE Access.*

Johansson, U., & Gabrielsson, P. (2019). Are Traditional Neural Networks Well-Calibrated? *International Joint Conference on Neural Networks (IJCNN).* Budapest, Hungary.

Johansson, U., Linusson, H., Löfström, T., & Boström, H. (2017). Model-agnostic nonconformity functions for conformal classification. *International Joint Conference on Neural Networks (IJCNN).* Anchorage, AK.

Kaakai, F., Adibhatla, S. "., Pai, G., & Escorihuela, E. (2023). Data-Centric Operational Design Domain Characterization for Machine Learning-Based Aeronautical Products. *Computer Safety, Reliability, and Security: 42nd International Conference, SAFECOMP.*

Kaakai, F., Adibhatla, S., Pai, G., & Escorihuela, E. (2023). Data-Centric Operational Design Domain Characterization for Machine Learning-Based Aeronautical Products. *Computer Safety, Reliability, and Security, 42nd SAFECOMP.* Toulouse, France.

Kabudi, T., Pappas, I., & Olsen, D. H. (2021). AI-enabled adaptive learning systems: A systematic mapping of the literature. *Computers and Education: Artificial Intelligence, 2*.

Kakaletsis, E., Symeonidis, C., Tzelepi, M., Mademlis, I., Tefas, A., Nikalaidis, N., & Pitas, I. (2022). Computer Vision for Autonomous UAV Flight Safety: An Overview and a Vision-based Safe Landing Pipeline Example. *Association for Computing Machinery, 54*(9).

Kaplan, A. D., Kessler, T. T., Brill, J. C., & Hancock, P. A. (2023). Trust in Artificial Intelligence: Meta-Analytic Findings. *Human Factors, 65*(2), 337-359.

Kaplan, A. D., Kessler, T. T., Brill, J. C., & Hancock, P. A. (2023). Trust in artificial intelligence: Meta-analytic findings. *Human Factors, 65*(2), 337-359.

Katz, G., Huang, D. A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., . . . Barrett, C. (2019). The Marabou Framework for Verification and Analysis of Deep Neural Networks. *Computer Aided Verification. CAV*.

Kläs, M., & Vollmer, A. M. (2018). Uncertainty in Machine Learning Applications: A Practice-Driven Classification of Uncertainty. *Computer Safety, Reliability, and Security. SAFECOMP 2018. Lecture Notes in Computer Science, 11094*.

Kläs, M., Vollmer, A.M. (2018). Uncertainty in Machine Learning Applications: A Practice-Driven Classification of Uncertainty. *Computer Safety, Reliability, and Security. SAFECOMP 2018. Lecture Notes in Computer Science, 11094*.

Koch, J. (2017). Design implications for Designing with a Collaborative AI. *AAAI Spring Symposium.*

Kristiadi, A., Hein, M., & Hennig, P. (2021). Learnable uncertainty under Laplace approximations. *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence (PMLR) vol 161.*

Kumar, A., Levine, A., & Feizi, S. (2021). Policy Smoothing for Provably Robust Reinforcement Learning. *arXiv:2106.11420.*

Kumar, R. S., O'Brien, D., Albert, K., Viljöen, S., & Snover, J. (2019). Failure Modes in Machine Learning Systems. arXiv:1911.11034v1.

L. Chen, G. P. (2017). Rethinking Atrous Convolution for Semantic Image Segmentation. *arXiv:1706.05587.*

L. Harison, P. Saunders and J. Janowitz. (1994). *Artificial Intelligence with Applications for Aircraft.* Federal Aviation Administration.

Lakshminarayanan, B., Pritzel, A., & Blundell, C. (2017). Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. *31st Conference on Neural Information Processing Systems.* Long Beach, CA.

Lakshminarayanan, B., Pritzel, A., & Blundell, C. (2017). Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. *Advances in Neural Information Processing Systems 30 (NIPS).*

Laminar Research. (2021). *X-Plane 11 Flight Simulator*. Retrieved from https://www.x-plane.com/

Landers, M., & Doryab, A. (2023). Deep Reinforcement Learning Verification: A Survey. *ACM Computing Surveys, Vol. 55.*

LaValle, S. M. (2006). *Planning Algorithms.* Cambridge University Press.

Lazarus, C., Lopez, J. G., & Kochenderfer, M. J. (2020). Runtime Safety Assurance Using Reinforcement Learning. *AIAA/IEEE 39th Digital Avionics Systems Conference.*

Lecture 9: Object Detection and Image Segmentation. (2024). In *CS231n: Deep Learning for Computer Vision.* Stanford.

Lecun, Y., Cortes, C., & Burges, C. J. (n.d.). *The MNIST Database*. Retrieved 2024, from https://yann.lecun.org/exdb/mnist/index.html

Lee, C. W., Nayeer, N., Garcia, D. E., Agrawal, A., & Liu, B. (2020). Identifying the Operational Design Domain for an Automated Driving System through Assessed Risk. *IEEE Intelligent Vehicles Symposium.* Las Vegas.

Lee, R., Mengshoel, O. J., Saksena, A., Gardner, R., Genin, D., Silbermann, J., . . . Kochenderfer, M. J. (2020). Adaptive Stress Testing: Finding Likely Failure Events with Reinforcement Learning. *Journal of Artificial Intelligence Research, Vol 69.*

Liang, S., Li, Y., & Srikant, R. (2018). Enhancing The Reliability of Out-of-distribution Image Detection in Neural Networks. *6th International Conference on Learning Representations (ICLR).* Vancouver, BC, Canada.

Liang, Y., Sun, Y., Zheng, R., & Huang, F. (2022). Efficient Adversarial Training without Attacking: Worst-Case-Aware Robust Reinforcement Learning. *NeurIPS.*

Liu, C., Arnon, T., Lazarus, C., Strong, C., Barrett, C., & Kochenderfer, M. J. (2020). *Algorithms for Verifying Deep Neural Networks.* Foundations and Trends in Optimization.

Liu, C., Arnon, T., Lazarus, C., Strong, C., Barrett, C., & Kochenderfer, M. J. (2021). Algorithms for Verifying Deep Neural Networks. *Foundations and Trends in Optimization, Now.*

Liu, C., Cofer, D., & Osipychev, D. (2023). Verifying an Aircraft Collision Avoidance Neural Network with Marabou. *NASA Formal Methods.*

Liu, C., Cofer, D., & Osipychev, D. (2023). Verifying an Aircraft Collision Avoidance Neural Network with Marabou. *NASA Formal Methods.*

Lomuscio, A. (2023). VENUS: FORMAL VERIFICATION FOR NEURAL SYSTEMS. *AIR FORCE RESEARCH LABORATORY, AFRL-RI-RS-TR-2023-152.*

Lopez, D. M., Althoff, M., Forets, M., Johnson, T. T., Ladner, T., & Schilling, C. (2023). ARCH-COMP23 Category Report: Artificial Intelligence and Neural Network Control Systems (AINNCS) for Continuous and Hybrid Systems Plants. *Proceedings of 10th International Workshop on Applied Verification of Continuous and Hybrid Systems.*

Luo, Y., & Tengyu, M. (2021). Learning Barrier Certificates: Towards Safe Reinforcement Learning with Zero Training Time violations. *35th Conference on Neural Information Processing Systems.*

Lutjens, B., Everett, M., & How, J. P. (2019). Certified Adversarial Robustness for Deep Reinforcement Learning. *3rd Conference on Robot Learning.* Osaka, Japan.

Maleki, F., Muthukrishnan, N., Ovens, K., Reinhold, C., & Forghani, R. (2020). Machine Learning Algorithm Validation: From Essentials to Advanced Applications and Implications for Regulatory Certification and Deployment. *Neuroimaging Clinics of North America, 30*(4), 433-445.

Maleki, Farhad et al. (2020). Machine Learning Algorithm Validation. *30*(4), 433-445.

Manokhim, V. (2023). *Practical Guide to Applied Conformal Prediction in Python.* Packt Publishing.

Manokhim, Valery. (2023). *Practical Guide to Applied Conformal Prediction in Python.* Packt Publishing.

Mathewson, K. W., & Pilarski, P. M. (2022). A Brief Guide to Designing and Evaluating Human-Centered Interactive Machine Learning. *ICLR.*

Matlab Toolbox for Neural Network Verification. (n.d.). Retrieved September 2023, from https://github.com/verivital/nnv

Mayer, R. C., Davis, J. H., & Schoorman, F. D. (1995). An Integrative Model of Organizational Trust. *The Academy of Management Review, 20*(3), 709-734.

Mehmood, U., Sheikhi, S., Bak, S., Smolka, S. A., & Stoller, S. D. (2022). The Black-Box Simplex Architecture for Runtime Assurance of Autonomous CPS. *NASA Formal Methods.*

Microsoft. (2021, April). *AirSim Simulator*. Retrieved from https://github.com/microsoft/AirSim

MIT LL Airspace Encounter Models. (n.d.). Retrieved 2023, from https://github.com/Airspace-Encounter-Models

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. arXiv:1312.5602v1. Retrieved from https://arxiv.org/abs/1312.5602

Modares, A., Sadati, N., Esmaeili, B., Yaghmaie, F. A., & Modares, H. (2023). Safe Reinforcement Learning via a Model-Free Safety Certifier,. *IEEE Transactions on Neural Networks and Learning Systems.*

Mohseni, S., Jagadeesh, A., & Wang, Z. (2019, May). *Predicting Model Failure using Saliency Maps in Autonomous Driving Systems.* Retrieved from https://doi.org/10.48550/arXiv.1905.07679

Murphy, K. P. (2022). Ch 2. Probability: Univariate Model. In *Probabilistic Machine Learning: An Introduction.* Cambridge: MIT Press.

Murphy, K. P. (2023). Probabilistic machine learning: Advanced topics. In *Chapter 33.* MIT Press.

Mussau, P., Hamilton, N., Lopez, D. M., Robinette, P., & Johnson, T. T. (2022). On Using Real-Time Reachability for the Safety Assurance of Machine Learning Controllers. *IEEE International Conference on Assured Autonomy (ICAA).*

Naidu, G., Zuva, T., & Sibanda, E. M. (2023). A Review of Evaluation Metrics in Machine Learning Algorithms. *Artificial Intelligence Application in Networks and Systems. CSOC, vol 724.*

NASA. (2021). *NASA Research Announcement NNH20ZEA001N-SWS: Vision 2045 of V&V for Autonomy.* NASA.

NASA. (2024). Retrieved from fmdtools 2.3.3 Documentation: https://nasa.github.io/fmdtools/

Nasrudin, M., Iriawan, N., Fithriasari, K., Pravitasari, A. A., & Hidayat, T. (2022, August). On the YOLOv4 Architecture for Fast and Real Time Congenital Heart Disease Detection Via Ultrasound Videos. *Matematika, 38*(2), 103-114. doi:https://doi.org/10.11113/matematika.v38.n2.1373

National Academies of Sciences, Engineering, and Medicine. (2022). *Human-AI Teaming: State of the Art and Research Needs.* Washingtion DC: The National Academies.

Nemani, V., Biggio, L., Huan, X., Hu, Z., Fink, O., Tran, A., . . . Hu, C. (2023). Uncertainty quantification in machine learning for engineering design and health prognostics: A tutorial. *Mechanical Systems and Signal Processing, 205*.

Newton H. Campbell, Michael J. Acheson and Irene M. Gregory. (2021). Dynamic Vehicle Assessment for Intelligent Contingency Management of Urban Air Mobility Vehicles. *AIAA SCITECH.* VIRTUAL EVENT.

Nikko, E., Sjanic, Z., & Heintz, F. (2021). Towards Verification and Validation of Reinforcement Learning in Safety-Critical Systems A Position Paper from the Aerospace Industry. *IJCAI.*

Nixon, J., Dusenberry, M., Ghassen, J., Nguyen, T., Liu, J., Zhang, L., & Tran, D. (2020). *Measuring Calibration in Deep Learning.* arXiv:1904.01685.

Nixon, J., Dusenberry, M., Jerfel, G., Nguyen, T., Liu, J., Zhang, L., & Tran, D. (2020). *Measuring Calibration in Deep Learning.* arXiv:1904.01685.

nnenum- The Neural Network Enumeration Tool. (n.d.). Retrieved September 2023, from https://github.com/stanleybak/nnenum

NVIDIA. (2018-2026). *Accelerating Inference in TensorFlow with TensorRT User Guide*. Retrieved 2022, from NVIDIA Docs Hub: https://docs.nvidia.com/deeplearning/frameworks/tf-trt-user-guide/index.html

Open Source Robotics Foundation. (2021, April). *Gazebo simulator*. Retrieved from http://gazebosim.org/

OpenAI. (2018). *Part 2: Kinds of RL Algorithms*. Retrieved 2023, from OpenAI Spinning Up: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

Parasuraman, R., Sheridan, T. B., & Wickens, C. D. (2000). A model for types and levels of human interaction with automation. *IEEE Transactions on systems, man, and cybernetics-Part A: Systems and Humans, 30*(3), 286-297.

Perez, T., & Courts, J. (2023). *BDR No. K-23001, Data-Driven Verification of Autonomous System Behaviors.* Boeing Development Report.

Perrotin, E., Roy, M., Herbulot, A., Devy, M., & Bousquet, F. (2022). Towards the certification of vision based systems: modular architecture for airport line detection. *11th European Congress on Embedded Real-Time Systems.* Toulouse, France.

Phillips, J., Hahn, C., Fontana, P., Broniatowski, D., & Przybocki, M. (2020). *Four Principles of Explainable Artificial Intelligence - Draft NISTIR 8312.* National Institute of Standards and Technology.

Pintz, M., Sicking, J., Poretschkin, M., & Akila, M. (2022). A Survey on Uncertainty Toolkits For Deep Learning. *ML Evaluation Standards Workshop at ICLR.*

Platt, J. (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers, 10*(3), 61-74.

Polyzotis, N., Zinkevich, M., Roy, S., Breck, E., & Whang, S. (2019). Data Validation for Machine Learning. *Proceedings of Machine Learning and Systems.*

Prashin Sharma. (2022). *UAS Contingency Management Autonomy with Experimentally Validated Models.* Ann Arbor: Ph.D. Thesis, University of Michigan.

Puterman, M. L. (2014). *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons.

PyTorch. (n.d.). *PyTorch documentation*, 2.9.1. Retrieved from https://docs.pytorch.org/docs/2.9/

Qi, Y., Conmy, P. R., Huang, W., Zhao, X., & Huang, X. (2022). A hierarchical HAZOP-like safety analysis for learning-enabled systems. *IJCAI-ECAI Workshop on Artificial Intelligence Safety.* Vienna.

R. Mur-Artal, J. M. (2015). ORB-SLAM: A Versatile and Accurate Monocular SLAM System,. *IEEE Transactions on Robotics*.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., & Dormann, N. (2021). Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research.*

Rahman, Q. M., Corke, P., & Dayoub, F. (2021). Run-Time Monitoring of Machine Learning for Robotic Perception: A Survey of Emerging Trends. *IEEE Access, 9*, 20067-20075.

Ramasamy, S., Sabatini, R., & Gardi, A. (2014). Avionics sensor fusion for small size unmanned aircraft Sense-and-Avoid. *IEEE Metrology for Aerospace.* Benevento, Italy.

Raschka, S. (2020). *Model Evaluation, Model Selection, and Algorithm.* arXiv:1811.12808.

Raz, A. K., Nolan, S. M., Levin, W., Mall, K., Mia, A., Mockus, L., . . . Williams, K. (2022). Test and Evaluation of Reinforcement Learning via Robustness testing and Explainable AI for High-Speed Aerospace Vehicles. *IEEE Aerospace Conference (AERO).*

Razzaghi, P., Tabrizian, A., Guo, W., Chen, S., Taye, A., Thompson, E., . . . Wei, P. (2022). A Survey on Reinforcement Learning in Aviation Applications. *arXiv:2211.02147v2.*

Recht, B. (2019). A Tour of Reinforcement Learning: The View from Continuous Control. *Annual Review of Control, Robotics, and Autonomous Systems.*

Riccio, V., Jahangirova, G., Stocco, A., Humbatova, N., Weiss, M., & Tonella, P. (2020). Testing machine learning based systems: a systematic mapping. *Empirical Software Engineering, 25*, 5193–5254.

Richter, C., & Roy, N. (2017). Safe Visual Navigation via Deep Learning and Novelty Detection. *Proceedings of Robotics: Science and Systems.* Cambridge, Massachusetts.

Rierson, L. (2017). Developing safety-critical software: a practical guide for aviation software and DO-178C compliance. *CRC Press.*

Ronghui Mu, Wenjie Ruan, Leandro Soriano Marcolino, Gaojie Jin, Qiang Ni. (2023). Certified Policy Smoothing for Cooperative Multi-Agent Reinforcement Learning. *The Thirty-Seventh AAAI Conference on Artificial Intelligence.*

Rosenhow, D., Lee, T. T., & Li, Z. X. (2022). Next-generation air routing: Integrating AI, multi-objective optimization, and collaborative decision making for efficient and sustainable flight planning. *International Journal of Enterprise Modelling, 16*(3), 136-144.

RTCA, Inc. (2000, April 19). DO-254: Design assurance guidance for airborne electronic hardware.

RTCA, Inc. (2011, December 13). RTCA DO-178C: Software Considerations in Airborne Systems and Equipment Certification.

RTCA, Inc. (2021, March 18). Minimum Operational Performance Standards (MOPS) for Detect and Avoid (DAA) Systems. Washington, DC.

S. Beland, I. Chang, A. Chen, M. Moser, J. Paunicka, D. Stuart, J. Vian, C. Westover, H. Yu. (2020). Towards Assurance Evaluation of Autonomous Systems. *International Conference on Computer-Aided Design (ICCAD).*

S. Ren, K. H. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *Twenty-ninth Conference on Neural Information Processing Systems.*

SAE. (2020). *AVSC00002202004 : AVSC Best Practice for Describing an Operational Design Domain: Conceptual Framework and Lexicon.* SAE ITC.

SAE. (2021). *AIR6988 Artificial Intelligence in Aeronautical Systems: Statement of Concerns.* SAE.

SAE. (2021). *AIR6988 Artificial Intelligency in Aeronautical Systems: Statements of Concerns.* SAE.

SAE. (2021). *J3016: Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles.* SAE.

SAE. (2023). *ARP-4761 A, Guidelines for Conducting the Safety Assessment Process On Civil Aircraft, Systems and Equipment.* SAE.

SAE G-34 / EUROCAE WG-114. (2021). *AIR6988: Artificial Intelligence in Aeronautical Systems: Statement of Concerns.* SAE.

SAE International. (2010, December 20). ARP4754A - Guidelines for Development of Civil Aircraft and Systems. doi:https://doi.org/10.4271/ARP4754A

Sanders-Reed, J. N. (2021). *Principles of Vision Enabled Autonomous Flight.* SPIE.

Saswata, P., Prince, D., Iyer, N., Durling, M., Visnevski, N., & Meng, B. (2023). Assurance of Machine Learning-Based Aerospace Systems: Towards an Overarching Properties-Driven Approach. *DOT/FAA/TC-23/54.*

Schulman, J., Levine, S., Moritz, P., Jordian, M. I., & Abbeel, P. (2017). Trust Region Policy Optimization. arXiv:1502.05477v5.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. arXiv:1707.06347v2.

Schumann, J., Mahadevan, N., Sweet, A., Bajwa, A. R., Lowry, M., & Karsai, G. (2019). Model-based System Health Management and Contingency Planning for Autonomous UAS. *AIAA SCITECH.* San Diego, CA.

Selby, C., & Staley, E. (2022). A Decision Metric for the Use of a Deep Reinforcement Learning Policy. *ICML Workshop on Responsible Decision Making in Dynamic Environments.* Baltimore, Maryland.

Selim, M., Alanwar, A., Shreyas, K., Gao, G., Pavone, M., & Johansson, K. H. (2022). Safe Reinforcement Learning Using Black-Box Reachability Analysis. *IEEE Robotics and Automation Letters, Vol 7.*

Seo, T., Sahoo, P. P., & Vamvoudakis, K. G. (2023). Verification of Adversarially Robust Reinforcement Learning Mechanisms in Aerospace Systems. *AIAA SCITECH.*

Shangding Gu*a*, Long Yang, Yali Du, Guang Chen, Florian Walter, Jun Wang, Yaodong Yang, Alois Knoll. (2023). A Review of Safe Reinforcement Learning: Methods, Theory and Applications. *arXiv:2205.10330v4.*

Shao, Y. S., Chen, C., Kousik, S., & Vasudevan, R. (2021). Reachability-Based Trajectory Safeguard (RTS): A Safe and Fast Reinforcement Learning Safety Layer for Continuous Control. *IEEE Robotics and Automation Letters Vol 6.*

Sharifi, I., Yildirim, M., & Fallah, S. (2023). Towards Safe Autonomous Driving Policies using a Neuro-Symbolic Deep Reinforcement Learning Approach. *arXiv:2307.01316v2.*

Sharma, P., Kraske, B., Kim, J., Laouar, Z., Sunberg, Z., & Atkins, E. (2023). Investigation of risk-aware MDP and POMDP contingency management autonomy for UAS. *Journal of Aerospace Information Systems, AIAA, (Submittted)*.

Shen, M., Yuheng, B., Sattigeri, P., Ghosh, S., Das, S., & Wornell, G. (2023). Post-hoc uncertainty learning using a dirichlet meta-model. *37th AAAI Conference on Artificial Intelligence.* Washington DC.

Shim, J., Park, J., Song, N. C., Jang, J., Choi, J.-Y., Lee, G., . . . Choi, S. (2023). Real-Time Optimal Route Planning by Deep Reinforcement Learning and Validation with Flight Test. *AIAA Aviation.* San Diego, CA.

Skoog, M. A., Hook, L. R., & Ryan, W. (2020). Leveraging ASTM Industry Standard F3269-17 for Providing Safe Operations of a Highly Autonomous Aircraft. *IEEE Aerospace Conference.* Big Sky, MT.

Smith, R. (2013). *Uncertainty Quantification: Theory, Implementation, and Applications, Second Edition.* Philadelphia, PA: Society for Industrial and Applied Mathematics.

Stefani, T., Girija, A. A., Mut, R., Hallerbach, S., & Krüger, T. (2023). FROM THE CONCEPT OF OPERATIONS TOWARDS AN OPERATIONAL DESIGN DOMAIN FOR SAFE AI IN AVIATION.

Sun, C., Cui, Y., Ning, M., Lu, Y., Cao, D., & Khajepour, A. (2024). Extending Operational Design Domain for Perception Systems Through Robust Learning. *IEEE Transactions on Intelligent Vehicles*, 1-11.

Sun, Y., Zheng, R., Hazzanzadeh, P., & Liang, Y. (2022). Certifiably Robust Multi-Agent Reinforcement Learning against adversarial communication. *ICML Workshop on Responsible Decision Making in Dynamic Environments.* Baltimore, Maryland.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction.* MIT Press.

Szeliski, R. (2022). In *Computer vision: algorithms and applications, Second Edition* (pp. 291-295). Springer Nature.

Szeliski, R. (2022). Ch2. Image Formation. In *Computer Vision: Algorithms and Applications, 2nd ed.* Springer.

T. Brown, B. M.-V. (2020). Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems.*

T. Lin, P. D. (2017). Feature Pyramid Networks for Object Detection. *Conference on Computer Vision and Pattern Recognition (CVPR),.* IEEE.

T. Lin, P. G. (2017). Focal Loss for Dense Object Detection. *International Conference on Computer Vision (ICCV),.* IEEE.

Taiga, A., Buchanan, E. K., Pleiss, G., Zemel, R., & Cunningham, J. P. (2022). Deep Ensembles Work, But Are They Necessary? *36th Conference on Neural Information Processing Systems (NeurIPS).* New Orleans.

Tambon, F., Majdinasab, V., Nikanjam, A., Khomh, F., & Antoniol, G. (2023). Mutation Testing of Deep Reinforcement Learning Based on Real Faults. *IEEE Conference on Software Testing, Verification and Validation.*

Tanay Kondapalli, C. P., Vaibhav, V., Konda, K. R., Praveen, K., & Kondoju, B. (2021). Real-time rain severity detection for autonomous driving applications. *IEEE Intelligent Vehicles Symposium.* Nagoya, Japan.

Tang, J., Lao, S., & Wan, Y. (2022). Systematic Review of Collision-Avoidance Approaches for Unmanned Aerial Vehicles. *IEEE Systems Journal, 16*(3), 4356-4367.

Tang, X., Yang, K., Wang, H., Jiahang, W., Qin, Y., Yu, W., & Cao, D. (2022). Prediction-Uncertainty-Aware Decision-Making for Autonomous Vehicles. *IEEE Transactions on Intelligent Vehicles, 7*(4), 849-862.

Tang, Xiaolin and Yang, Kai and Wang, Hong and Wu, Jiahang and Qin, Yechen and Yu, Wenhao and Cao, Dongpu. (2022). Prediction-Uncertainty-Aware Decision-Making for Autonomous Vehicles. *IEEE Transactions on Intelligent Vehicles, 7*(4), 849-862.

TenorFlow Lite Conversion. (2022). Retrieved from https://www.tensorflow.org/lite/models/convert/

TensorFlow Model Optimization. (n.d.). Retrieved 2022, from https://www.tensorflow.org/lite/performance/model_optimization

TensorFlow YOLOv4. (2020, August). Retrieved 2022, from https://github.com/hunglc007/tensorflow-yolov4-tflite

Teomitzi, H. E., & Schmidt, J. R. (2021). Concept and Requirements for an Integrated Contingency Management Framework in UAS Missions. *IEEE Aerospace Conference.* Big Sky, MT.

The Grand Challenge. (2004, March 13). Retrieved from Defense Advanced Research Projects Agency: https://www.darpa.mil/about-us/timeline/-grand-challenge-for-autonomous-vehicles.

Tibshirani, R. J., Barber, R. F., Candes, E., & Ramdas, A. (2019). Conformal Prediction Under Covariate Shift. *Advances in Neural Information Processing Systems 32.* Vancouver, Canada.

Torens, C., Juenger, F., Schirmer, S., Schopferer, S., Khukov, D., & Dauer, J. C. (2023). Ensuring Safety of Machine Learning Components Using Operational Design Domain. *AIAA SCITECH.* National Harbor, MD.

Torens, C., Juenger, F., Schirmer, S., Schopferer, S., Maienschein, T. D., & Dauer, J. C. (2022). Machine Learning Verification and Safety for Unmanned Aircraft - A Literature Study. *AIAA SCITECH.* San Diego, CA.

Torens, C., Juenger, F., Schirmer, S., Schopferer, S., Zhukov, D., & Dauer, J. C. (2023). Ensuring Safety of Machine Learning Components Using Operational Design Domain. *AIAA SCITECH Forum.* National Harbor, MD.

Torfah, H., Xie, C., Junges, S., Vazquez-Chanlatte, M., & Seshia, S. A. (2022). Learning Monitorable Operational Design Domains for Assured Autonomy. *Automated Technology for Verification and Analysis (ATVA)*, 3-22.

Tran, D., Lakshminarayanan, B., & Snoek, J. (2020). Practical Uncertainty Estimation and Out-of-Distribution Robustness in Deep Learning. *Neurips.*

Triva, J., Grbić, R., Vranješ, M., & Teslić, N. (2022). Weather Condition Classification in Vehicle Environment Based on Front-View Camera Images. *21st International Symposium INFOTEH-JAHORINA (INFOTEH).* East Sarajevo, Bosnia and Herzegovina.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, Sergey Levine. (2018). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. arXiv:1801.01290v2.

U. Johansson, H. Linusson, T. Löfström and H. Boström. (2017). Model-agnostic nonconformity functions for conformal classification. *International Joint Conference on Neural Networks (IJCNN).* Anchorage, AK.

UL Standards & Engagement. (2019). *ANSI/UL 4600 Standard for Safety for the Evaluation of Autonomous Products.* ANSI.

Usach, H., Torens, C., Adolf, F., & Vila, J. (2017). Architectural Considerations Towards Automated Contingency Management for Unmanned Aircraft. *AIAA Information Systems.* Grapevine, Texas.

van Wesel, P., & Goodloe, A. D. (2017). Challenges in the Verification of Reinforcement Learning Algorithms. *NASA/TM{2017{219628.*

Vidot, G., Gabreau, C., Ober, I., & Ober, I. (2021). *Certification of embedded systems based on Machine Learning: A survey*. Retrieved from https://doi.org/10.48550/arXiv.2106.07221

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, Koray Kavukcuoglu. (2016). *Asynchronous Methods for Deep Reinforcement Learning.* arXiv. doi:10.48550/arXiv.1602.01783

W. Guo and P. Wei. (2022). Explainable Deep Reinforcement Learning for Aircraft Separation Assurance. *IEEE/AIAA 41st Digital Avionics Systems Conference (DASC).* Portsmouth, VA.

W. Liu, D. A. (2016). SSD: Single Shot MultiBox Detector,. *European Conference on Computer Vision (ECCV*.

Walden, D. D., Shortell, T. M., Roedler, G. J., Delicado, B. A., Mornas, O., Yew-Seng, Y., & Endler, D. (2023). *INCOSE Systems Engineering Handbook: A guide for system life cycle processes and activities* (5 ed.). (INCOSE, Ed.) Wiley.

Walker, W. E., Harremoës, P., Rotmans, J., van der Sluijs, J. P., van Asselt, M., Jannsen, P., & Krayer von Krauss, M. P. (2003). Defining Uncertainty: A Conceptual Basis for Uncertainty Management in Model-Based Decision Support. *Integrated Assessment, 4*(1), 5-17.

Wang, H., & Ji, Q. (2024). Epistemic Uncertainty Quantification For Pre-Trained Neural Networks. *CVPR.* Seattle,WA.

Wang, Q., Ma, Y., & Zhao, K. (2020). A Comprehensive Survey of Loss Functions in Machine. *Annals of Data Science.*

Wang, Y., Zhan, S., Wang, Z., Huang, C., Wang, Z., Yang, Z., & Zhu, Q. (2023). Joint Differentiable Optimization and Verification for Certified Reinforcement Learning. *Association for Computing Machinery.*

Willems J.C. (2007). *The behavioral approach to open and interconnected systems.* IEEE Control Systems Magazine 27(6): 46-99.

Wu, F., Li, L., Huang, Z., Vorobeychik, Y., Zhao, D., & Li, B. (2022). CROP: Certifying Robust Policies for Reinforcement Learning Through Functional Smoothing. *ICLR.*

Wu, J., Sibai, H., & Vorobeychik, Y. (2022). Certifying Safety in Reinforcement Learning under Adversarial Perturbation Attacks. *arXiv:2212.14115v1.*

Xia, G., & Bouganis, C.-S. (2022). *On the Usefulness of Deep Ensemble Diversity for Out-of-Distribution Detection.* arXiv:2207.07517.

Xiang, Y., & Zhang, Y. (2015). Sense and avoid technologies with applications to unmanned aircraft systems: Review and prospects. *Progress in Aerospace Sciences, 74*, 152-166.

Xiao, H., Rasul, K., & Vollgraf, R. (2017). *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms.* arXiv:1708.07747.

Xu, M., Liu, Z., Huang, P., Ding, W., Cen, Z., Li, B., & Zhao, D. (2022). Trustworthy Reinforcement Learning Against Intrinsic Vulnerabilities: Robustness, Safety, and Generalizability. *arXiv:2209.08025v1.*

Y. He, H. Yu, G. Brat, M. Davies. (2021). Statistical learning framework for safety and failure analysis of a DNN-based autonomous aircraft system. *IEEE International Conference on Machine Learning and Applications (ICMLA).*

Y. Ma, Z. W. (2020). Artificial Intelligence Applications in the Development of Autonomous Vehicles: A Survey. *IEEE/CAA J. Autom. Sinica*, (pp. 315-329).

Yang, D., Ngoc, K. M., Shin, I., Lee, K.-H., & Hwang, M. (2021). Ensemble-Based Out-of-Distribution Detection. *Electronics, 10*(5).

Yang, J., Zhou, K., Li, Y., & Liu, Z. (2022). Generalized Out-of-Distribution Detection: A Survey. arXiv:2110.11334v2.

Yang, J., Zhou, K., Li, Y., & Liu, Z. (2024). Generalized Out-of-Distribution Detection: A Survey. *International Journal of Computer Vision*.

Zhang, C., Vinyals, O., Munos, R., & Bengio, S. (2018). A Study on Overfitting in Deep Reinforcement Learning. *arXiv:1804.06893v2.*

Zhang, H., Chen, H., Xiao, C., Li, B., Liu, M., Boning, D., & Hsieh, C.-J. (2020). Robust Deep Reinforcement Learning against Adversarial Perturbations on State Observations. *34th Conference on Neural Information Processing Systems.* Vancouver, Canada.

Zhang, R., Flathmann, C., Musick, G., Schelble, B., McNeese, N. J., Knijnenburg, B., & Duan, W. (2024). I Know This Looks Bad, But I Can Explain: Understanding When AI Should Explain Actions In Human-AI Teams. *Association for Computing Machinery, 14*(1).

Zhou, J., & Kwan, C. (2018). A High Performance Contingency Planning System for UAVs with Lost Communication. *IEEE International Conference on Prognostics and Health Management.* Seattle, WA.

A Comparison of different methods for uncertainty quantification

Table A- 1. Comparison of different methods for UQ

	Deep Ensemble	**Monte Carlo Dropout**	**Bayesian NN**	**Deep Evidential Regression**	**Test-Time Data Augmentation**
Description	Prediction and UQ at inference based on individual predictions received from multiple different networks	Predictions and UQ at inference are based on the weights of the hidden units scaled by the dropout probability	Model parameters are explicitly modeled as random variables. In a single model pass, parameters are sampled from this distribution	Network predicts the parameters of evidential distribution	Predictions and UQ at inference are based on single model and different forward passes of different augmented version of original input
Pros	Deep Ensemble increases the accuracy and provides a smooth uncertainty around the prediction	Easy Parallelization Training time is constant. Model Empirical statistics obtained from sampling different dropout configuration	Most accurate type of uncertainty Inference time complexity is linear	Computationally Cheap Calibrated uncertainty estimates	Low sensitivity to initialization and parameters of training process
Cons	Uncertainty estimate is usually overconfident. Linearly more expensive (both training and inference)	Noisy uncertainty around the prediction Uncertainty estimate is stochastic Inference time is linearly more expensive	Computationally expensive for training Uncertainty estimate is stochastic	Results are affected by choice of prior tuning of regularization coefficient	High computational effort at inference due to several data augmentations at test time