ECSS-E-HB-40-02A
15 November 2024

Space engineering

Machine learning handbook

ECSS Secretariat
ESA-ESTEC
Requirements & Standards Section
Noordwijk, The Netherlands

Foreword

This Handbook is one document of the series of ECSS Documents intended to be used as supporting material for ECSS Standards in space projects and applications. ECSS is a cooperative effort of the European Space Agency, national space agencies and European industry associations for the purpose of developing and maintaining common standards.

This handbook has been prepared by the ECSS-E-HB-40-02A Working Group, reviewed by the ECSS Executive Secretariat and approved by the ECSS Technical Authority.

Disclaimer

Published by: ESA Requirements and Standards Section
ESTEC, P.O. Box 299,
2200 AG Noordwijk
The Netherlands

Change log

ECSS-E-HB-40-02A 15 November 2024	First issue

Table of contents

Figures

Tables

1 Scope

1.1 Purpose

The Machine Learning Handbook provides guidelines on how to create reliable machine learning functions and perform the verification and validation considering the specifics of machine learning development practices.

Guidelines are provided for selecting, preparing, and validating data, as well as for training, testing, and applying machine learning models within a so-called 'safety cage' architecture. The handbook focused on data driven approaches with both supervised and unsupervised learning methods.

1.2 Executive Summary

1.2.1 AI & ML application in space domain

Breakthroughs in AI are enabling new paradigms, including in the field of software development which can be see a change from developers writing source code directly to an increased use of machine learning system that take a desired behaviour as goal and generate the relevant source code automatically. Further significant changes related to increasing utilization of AI to augment or even replace functions and systems are likely to affect a diverse set of areas such as engineering and manufacturing of space systems, operations and eventual knowledge discovery in data collected by space systems.

There are many exciting opportunities to consider the entire software development ecosystem and how it can be adapted to this new programming paradigm. Ideally AI algorithms can offer time and cost savings, as well as greater versatility and the ability to respond more intelligently to system behaviours not encountered during the training phase as long as it is within the training domain.

Although the working group members are working in space industry, they analysed the initial results from standardization activities on certification in aerospace and military, as they are well connected to the other domains and could bring in valuable guidelines and reference documentation.

Machine Learning AI is a revolution that will not disappear but will change the space software development world permanently.

Throughout the handbook, the AI taxonomy from [EASA Roadmap], as illustrated in Figure 1-1, is applied:

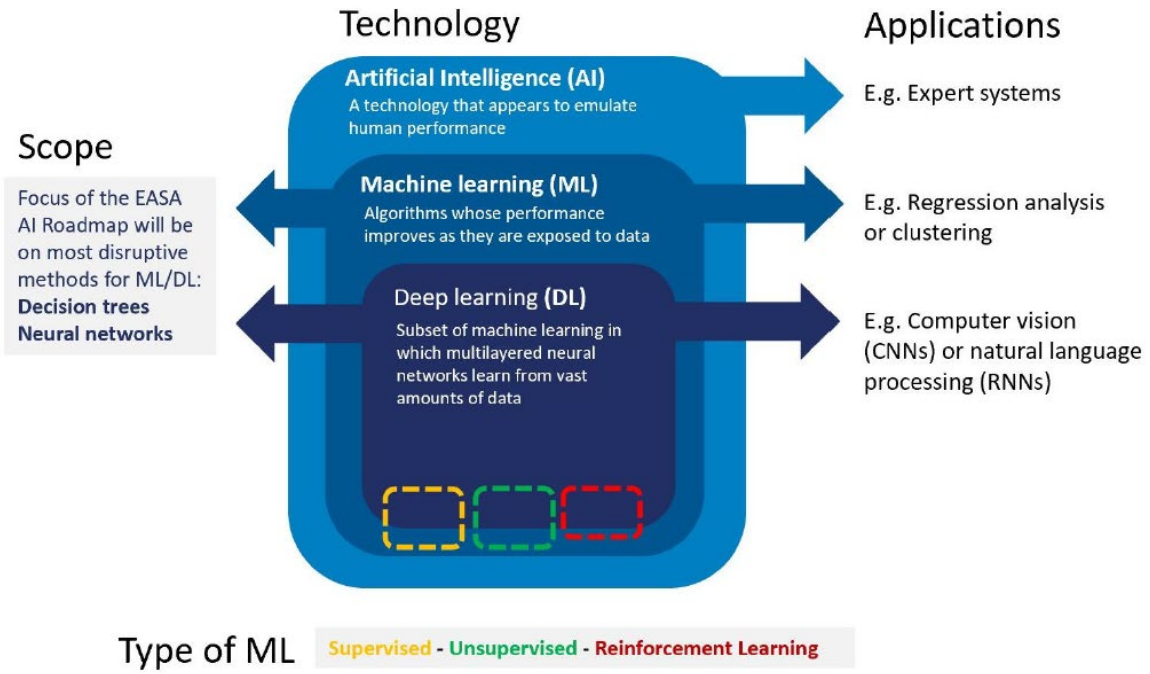

Figure 1-1: [EASA Roadmap] AI Taxonomy

1.2.2 AI @ ESA

AI is already intensively used in mostly all European space missions. More than 300 studies have been done or are in execution around AI applications in all domains (Science, Exploration, Navigation, Earth Observation, Technology & Engineering, Operations, Launchers and Telecommunication).

AI is the key enabler for higher levels of automation in data processing on ground and for autonomous operations for all future missions.

Different operational AI solutions exist, mainly on ground (e.g. for satellite image analysis or operations assistance) and first flight tests are done. NASA seems to be quite advanced for example at the Perseverance rover, which is heavily using AI.

An AI function is always part of a complete system, contributing to a hybrid solution that could involve a mix of manual programming, classical AI (symbolic AI), or machine learning modules. In the handbook also guidelines are provided on how to connect different kind of AI functions in a clever way, the safety cage architecture.

Like ESA also other space agencies do not yet have finalized AI engineering and qualification standardization, instead it is up to each project to define its own standards. This handbook it the first step to provide state-of-the-art guidelines to a broad community to ease AI applications at ESA.

1.3 Justification and Scope of the Handbook

Nowadays AI and specifically machine learning applications, cannot be verified and validated for critical applications based on standard requirements, as the ECSS requirements on software quality are not adapted to be applied for learning systems but for deterministic control algorithms. Likewise, there are a multitude of potential use-cases for AI within the space domain, e.g., health monitoring, AOCS, VBN, image processing and more, from engineering to exploitation, but the verification and validation of such developments has to consider application specific constraints.

1.3.1 Intended programs and target users

The ML Handbook can be used for AI software development in European Space Domain, except Cat A criticality.

AI will be a major driver for raising space systems autonomy especially for future exploration activities but also for reducing operations costs of LEO systems and for other autonomous/robotic elements.

Today the major use of AI is on ground, but major value will be generated by also using AI in flight systems and potentially in the frame of decision making.

This handbook is intended for any space programs which have the interest in utilizing AI and can be applied (and is already applied) in all space program domains:

- Exploration
- Navigation
- Earth Observation
- Launchers
- Telecommunication

2 References

ECSS-S-ST-00-01	ECSS System - Glossary of terms
ECSS-E-ST-40	Space engineering – Software
ECSS-Q-ST-30	Space product assurance - Dependability
E CSS-Q-ST-30-02	Space product assurance - Failure modes, effects (and criticality) analysis (FMEA/FMECA)"
ECSS-Q-ST-40	Space product assurance – Safety
ECSS-Q-ST-60-02	Space product assurance - ASIC and FPGA development
ECSS-Q-ST-80	Space product assurance - Software product assurance
ECSS-Q-HB-80-03	Space product assurance - Software dependability and safety handbook

ID	References	Link
[AI@ESA]	Artificial Intelligence in ESAv1.3	The document is available on request from ESA
[AVEHIC]	Manel Brini, Paul Crubille, Benjamin Lussier, Walter Schön. Validation of safety necessities for a Safety-Bag component in experimental autonomous vehicles. 14th European Dependable Computing Conference (EDCC), Sep 2018, Iasi, Romania. pp.33-40, 10.1109/EDCC.2018.00017	https://hal.archives-ouvertes.fr/hal-01998333
[DE-Standard RM]	German Standardisation Roadmap on Artificial Intelligence	https://www.din.de/resource/blob/772610/e96c34dd6b12900ea75b460538805349/normungsroadmap-en-data.pdf
[DEEL]	DEEL Whitepaper	https://arxiv.org/ftp/arxiv/papers/2103/2103.10529.pdf"
[DSG]	Data Safety Guidance	https://scsc.uk/r127F:2
[EASA Roadmap]	EASA AI Roadmap V1	https://www.coursehero.com/file/63688696/EASA-AI-Roadmap-v10pdf/
[EASA ConceptPaper]	EASA Concept Paper: First usable guidance for Level 1 machine learning applications – A deliverable of the EASA AI Roadmap	https://www.easa.europa.eu/en/downloads/126648/en
[ELEKTRA]	P. Klein, "The safety-bag expert system in the electronic railway interlocking system Elektra," Expert Systems with Applications, vol. 3, pp. 499 – 506, 1991.	
[ER-022/AIR6988]	19.ISE.OP.200 EDA Service contract for the Safe Autonomous Flight Termination (SAFE-Term) - Standardisation, Certification and Regulation Report	https://www.safeterm.eu/sites/default/files/2022-03/D2.5%20SAFETERM%20-%20Standardisation%20Certification%20and%20Regulation%20Report-Issue_3_4.pdf

ID	References	Link
[ESA-TECQQS-TN-022868]	Q80-Review	Technical Note: ESA-TECQQS-TN-022868. Machine Learning and Software Product Assurance: Bridging the Gap - YGT Report, 1.0, 2021 Jonathan Woodburn
[EUROCAE]	EUROCAE Statement of concerns Artificial Intelligence in Aeronautical Systems	https://eurocae.net/
[Goodfellow et al. 2016]	Deep Learning book	Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron: Deep Learning. The MIT Press, 2016. – ISBN 9780262035613
[Jain et al. 2020]	Jain et, "Overview and Importance of Data Quality for Machine Learning Tasks", Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2020	https://dl.acm.org/doi/proceedings/10.1145/3394486
[Kapoor et al. 2022]	Leakage and Reproducibility Crisis in ML-based Science	Leakage and the Reproducibility Crisis in ML-based Science
[Lones 2021]	ML pitfalls	How to avoid machine learning pitfalls: a guide for academic researchers
[MATHVERIF1]	Aditi Raghunathan, Jacob Steinhardt, Percy Liang, "Semidefinite relaxations for certifying robustness to adversarial examples" in Advances in Neural Information Processing Systems, pp. 10877-10887. 2018	https://doi.org/10.48550/arXiv.1811.01057
[MATHVERIF2]	Ruediger Ehlers, "Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks"	https://doi.org/10.48550/arXiv.1705.01320
[MATHVERIF3]	M. Fazlyab, M. Morari, and G. J. Pappas, "Probabilistic Verification and Reachability Analysis of Neural Networks via Semidefinite Programming," IEEE Conference on Decision and Control (CDC), 2019.	https://doi.org/10.48550/arXiv.1910.04249
[Ng et al.]	Structuring ML Projects	Coursera: Structuring Machine Learning Projects

ID	References	Link
[SAO]	Safety Assurance Objectives for Autonomous Systems	https://scsc.uk/r156A:1
[SCAGE]	Safety cage Architecture	http://safety.addalot.se/upload/2019/Safety_cage_SCSSS.PDF
[Software2.0]	Software 2.0	https://karpathy.medium.com/software-2-0-a64152b37c35
[SPAAS]	JP. Blanquart, S. Fleury, M. Hernek, C. Honvault, F. Ingrand, JC. Poncet, D. Powell, N. Strady-Lécubin, P. Thévenod. Software Product Assurance for Autonomy On-board Spacecraft. January 2003. Conference: DAta Systems In Aerospace.	https://www.researchgate.net/publication/256494270_Software_Product_Assurance_for_Autonomy_On-board_Spacecraft
[Vandewiele et al. 2022]	Overly optimistic prediction	Overly optimistic prediction results on imbalanced data: a case study of flaws and benefits when applying over-sampling

[RD-1]	Securing Machine Learning Algorithms – European Union Agency for Cybersecurity (ENISA)	https://www.enisa.europa.eu/publications/securing-machine-learning-algorithms/@@download/fullReport
[RD-2]	European Commission (2020) White Paper on Artificial Intelligence – a European approach to excellence and trust	https://digital-strategy.ec.europa.eu/en/consultations/white-paper-artificial-intelligence-european-approach-excellence-and-trust
[RD-3]	Special Committee on Artificial Intelligence in a Digital Age	https://www.europarl.europa.eu/cmsdata/246872/A9-0088_2022_EN.pdf
[RD-4]	NIST (USA) - (2021) Artificial Intelligence Risk Management Framework	https://www.regulations.gov/document/NIST-2021-0004-0001
[RD-5]	UK Ministry of Defence, policy paper: Defence Artificial Intelligence Strategy (June 2022)	https://www.gov.uk/government/publications/defence-artificial-intelligence-strategy
[RD-6]	Confiance.ai, France: French community to design and industrialise trustworthy AI-based critical systems.	https://www.confiance.ai/en/

	Including Airbus, Safran, Thales, and others, Launched April 2022. Focus on National level: characterisation of AI, trustworthy AI by design, data and knowledge engineering, mastering AI-based system engineering and trustworthy AI for embedded systems.	
[Gehr et al. 2018]	AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation"	T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri and M. Vechev, "AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation," 2018 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2018, pp. 3-18, doi: 10.1109/SP.2018.00058.
[Singh et al. 2019]	An abstract domain for certifying neural networks	Singh, G., Gehr, T., Püschel, M. and Vechev, M., 2019. An abstract domain for certifying neural networks. Proceedings of the ACM on Programming Languages, 3(POPL), pp.1-30
[Cheng et al. 2017]	Maximum Resilience of Artificial Neural Networks	Cheng, CH., Nührenberg, G., Ruess, H. (2017). Maximum Resilience of Artificial Neural Networks. In: D'Souza, D., Narayan Kumar, K. (eds) Automated Technology for Verification and Analysis. ATVA 2017. Lecture Notes in Computer Science(), vol 10482. Springer, Cham
[Au et al. 2014]	Engineering risk assessment with subset simulation	Au, S.K. and Wang, Y., 2014. Engineering risk assessment with subset simulation. John Wiley & Sons.
[Schwaiger et al. 2022]	A modular subset simulation toolbox for Matlab	Schwaiger, F., Shi, D., Mishra, C., Höhndorf, L. and Holzapfel, F., 2022. A modular subset simulation toolbox for Matlab. In AIAA Scitech 2022 Forum (p. 1893).
[Puppe 1993]	Systematic Introduction to Expert Systems.	Puppe, F. (1993). Systematic Introduction to Expert Systems. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-77971-8
[Sarker 2021]	Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions	https://link.springer.com/article/10.1007/s42979-021-00815-1
[neptune.ai]	Model-Based and Model-Free Reinforcement Learning: Pytennis Case Study	https://neptune.ai/blog/model-based-and-model-free-reinforcement-learning-pytennis-case-study
[INT-ML]	Interpretable Machine Learning	Interpretable Machine Learning (christophm.github.io)

[RD Miller2017]	Explanation in artificial intelligence: Insights from the social sciences.	Explanation in artificial intelligence: Insights from the social sciences - ScienceDirect
[Powers 2007]	Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation	https://www.flinders.edu.au/science_engineering/fms/School-CSEM/publications/tech_reps-research_artfcts/TRRA_2007.pdf
[Ying 2019]	An Overview of Overfitting and its Solutions	http://dx.doi.org/10.1088/1742-6596/1168/2/022022
[Sehra 2021]	Undecidability of Underfitting in Learning Algorithms	https://www.researchgate.net/publication/349106886_Undecidability_of_Underfitting_in_Learning_Algorithms

3
Terms, definitions and abbreviated terms

3.1 Terms from other documents

a. For the purpose of this document, the terms and definitions from ECSS-S-ST-00-01 apply.

b. For the purpose of this document, the terms and definitions from ECSS-E-ST-40 apply, in particular for the following term:

1. **validation <software>**
2. **verification <software>**

3.2 Terms specific to the present document

3.2.1 Accuracy (of the data)

degree of conformance between the estimated or measured value and its true value.

[Source: EASA, ISO 5725]

3.2.2 Artificial intelligence

technology that appears to emulate human performance typically by learning, coming to its own conclusions, appearing to understand complex content, engaging in natural dialogues with people, enhancing human cognitive performance (also known as cognitive computing) or replacing people on execution of non-routine tasks

[Source: EASA]

3.2.3 Data set

sample of data used for various development phases of the model, i.e. the model training, the learning process verification, and the inference model verification

[Source: EASA]

3.2.4 Data set (test data)

data used to assess the performance of the model, independent of the training data set

[Source: EASA]

3.2.5 Data set (training data)

data that is input to an ML model in order to establish its behaviour

[Source: EASA]

3.2.6 Data set (validation data)

Data used to tune a subset of the hyper-parameters of a model (e.g. number of hidden layers, learning rate, etc.).

[Source: EASA]

3.2.7 Explainability

This guidance makes a clear distinction between two types of explainability driven by the profile of the user and their needs:

- The information required to make a machine learning model understandable; and
- Understandable information for the user on how the systems came to its results.

[Source: EASA]

3.2.8 Evaluation (of the model)

Model Evaluation is the process of assessing a machine learning model's performance by comparing its outputs to predefined performance metrics such as accuracy, recall, precision, mean squared error, and others. This process is focused on quantitative measurements that help determine the model's effectiveness in performing its designated tasks.

For instance, a common metrics used for the classification task is the F1 score (harmonic mean between precision and recall), ranging between 0 and 1. The acceptability level is highly dependent on the application, how critical the ML algorithm is, and its context. Yet, as a rule of thumb, practitioner should seek a value greater than 0.9 to consider the algorithm has very good performance, and discard algorithms whose F1-score is lower than 0.8.

3.2.9 Feature (in computer science)

any piece of information which is relevant for solving the computational task related to a certain application

- Feature (in machine learning in general) — A feature is an individual measurable property or characteristic of a phenomenon being observed.
- Feature (in computer vision) — A feature is a piece of information about the content of an image; typically, about whether a certain region of the image has certain properties.

[Source: EASA]

3.2.10 Hyperparameter

parameter that is used to control the algorithm's behaviour during the learning process (e.g., for deep learning with neural networks, the learning rate, the batch size or the initialisation strategy).

NOTE *Hyper-parameters affect the time and memory cost of running the algorithm, or the quality of the model obtained at the end of the training process. By contrast, other parameters, such as node weights or biases, are the result of the training process.*

[Source: EASA]

3.2.11 Inference

process of feeding the machine learning model an input and computing its output

NOTE *See also related definition of Training.*

[Source: EASA]

3.2.12 Intended functionality

behaviour specified for a system

[Source: ISO]

3.2.13 Label

In the context of machine learning, the target variable assigned to a sample

[Source: ISO]

3.2.14 Machine learning

branch of AI concerned with the development of algorithms that allow computers to evolve behaviours based on observing data and making inferences on this data

[Source: EASA]

3.2.15 Machine learning (supervised)

process of learning in which the ML algorithm processes the input data set, and a cost function measures the difference between the ML model output and the labelled data. The ML algorithm then adjusts the parameters to increase the accuracy of the ML model.

[Source: EASA]

3.2.16 Machine learning (unsupervised)

process of learning in which the ML algorithm processes the data set, and a cost function indicates whether the ML model has converged into a stable solution. The ML algorithm then adjusts the parameters to increase the accuracy of the ML model.

[Source: EASA]

3.2.17 Machine learning (reinforcement)

process of learning in which the agent(s) is (are) rewarded positively or negatively based on the effect of the actions on the environment. The ML model parameters are updated from this trial-and-error sequence to optimise the outcome.

[Source: EASA]

3.2.18 Machine learning inference model

The ML model obtained after transformation of the trained model, so that the model is adapted to the target platform.

[Source: EASA]

3.2.19 Machine learning model

parameterised function that maps inputs to outputs. The parameters are determined during the training process.

[Source: EASA]

3.2.20 Machine learning trained model

ML model which is obtained at the end of the learning/training phase

[Source: EASA]

3.2.21 Offline learning

The process of learning where the ML model is frozen at the end of the development phase;

[Source: EASA]

3.2.22 Online learning

The process of learning where the ML model parameters can be updated based on data acquired during operation (see also adaptivity).

[Source: EASA]

3.2.23 Overfitting

creating a model that matches the training data so closely that the model fails to make correct predictions on new data. Overfitting can be caused by characteristics of the training set (size of dataset, relation of training set distribution relative to distribution in population), characteristics of the model (tendency to create high bias or variance errors) and/or aspects of the training (number of iterations, utilization of regularization or other mitigating options).

[Source: Ying 2019]

3.2.24 Precision

metric for classification models, also known as positive predictive value. Precision identifies the frequency with which a model was correct when predicting the positive class. In other words, it is the fraction of relevant instances among the retrieved instances

[Source: Powers 2007]

3.2.25 Prediction (predictability)

degree to which a correct forecast of a system's state can be made quantitatively

NOTE *Limitations on predictability could be caused by factors such as a lack of information or excessive complexity.*

[Source: EASA]

3.2.26 Reliability

probability that an item will perform a required function under specified conditions, without failure, for a specified period of time.

3.2.27 Resilience

In the context of this guidance, the resilience definition is derived from DEEL White Paper on Machine learning in Certified System (DEEL Certification Workgroup, 2021) where resilience is defined as the ability of a system to continue to operate while an error or a fault has occurred.

[Source: EASA]

3.2.28 Robustness

For an input varying in a region of the input state space, a system producing expected outputs.

In [DEEL] White Paper on Machine learning in Certified System (DEEL Certification Workgroup, 2021), robustness is defined as the ability of the system to perform the intended function in the presence of abnormal or unknown inputs, and to provide equivalent response within the neighbourhood of an input.

[Source: EASA]

3.2.29 Safety of the intended functionality (SOTIF)

Absence of unreasonable risk due to hazards resulting from functional insufficiencies of the intended functionality or from reasonably foreseeable misuse by persons. Nominal performance includes intended functionality and the implementation of intended functionality that can be affected by performance limitations or by foreseeable misuse by persons.

[Source: ISO]

3.2.30 Test case

Set of conditions to determine if a system is working according to its intended functionality

[Source: ISO]

3.2.31 Training

The process of setting appropriate weights for a machine learning model, via optimization. For example, in supervised learning the training data consists of input (e.g. an image) / output (e.g. a class label) pairs and the ML model 'learns' the function that maps the input to the output, by optimising its internal parameters. See also the related definition of Inference.

[Source: EASA]

3.2.32 Underfitting

Producing a model with poor predictive ability because the model hasn't captured the complexity of the training data. Many problems can cause underfitting, including:

- Training on the wrong or incomplete set of features.
- Training for too few epochs or at too low a learning rate.
- Training with too high a regularization rate.
- Providing too few hidden layers in a deep neural network.

[Source: Sehra 2021]

3.2.33 Verification

This process is intended to confirm that adequate specifications for the data product and the machine learning model product exist, and that the machine learning software product outputs are interchangeable and consistent with outputs of the machine learning model product trained with the data product.

3.2.34 Validation

This process is intended to confirm that the data and model quality properties and baseline requirements are completely implemented in the data product, machine learning model product, and machine learning-based system product.

3.3 Abbreviated terms

For the purpose of this Standard, the abbreviated terms and symbols from ECSS-S-ST-00-01 and the following apply:

Abbreviation	Meaning
AI	Artificial Intelligence
AOCS	Attitude and Orbit Control System
ASIC	Application-Specific Integrated Circuit
ConOps	Concept of Operations
CPU	Central Processing Unit
EDA	Exploratory data analysis
FDIR	Fault detection and isolation
FMEA	Failure mode and effects analysis

Abbreviation	Meaning
FMECA	Failure mode, effects, and criticality analysis
FPGA	Field-programmable gate array
GPU	Graphics processing unit
HazOp	Hazard and operability study
HSIA	Hardware Software Interaction Analysis
LEO	Low Earth Orbit
LIDAR	Laser imaging, detection, and ranging
LIME	Local surrogate models
LSIs	Less significant institutions
MBSE	Model Based System Engineering
ML	Machine Learning
NN	Neural Network
ODD	Operational Design Domain
OOD	Out-Of-Distribution
QA	Quality Assurance
ReLU	Rectified linear unit
RL	Reinforcement Learning
ROI	Return-Of-Investment
SAE	Sparse Autoencoders
SEU	Single Event Upset
SFMEA	Software Failure Modes and Effects Analysis
SLT	Statistical learning theory
SOTIF	Safety of the intended functionality
SWOT	Strengths, Weaknesses, Opportunities, and Threats
UML	Unified Modeling Language
V&V	Verification and validation
VBN	Vision Based Navigation
XAI	Explainable Artificial Intelligence

4 Overview

4.1 Perimeter and objectives

The Handbook recommends guidelines applicable to the machine learning development process, encompassing data gathering, data processing, training, model testing. The concept of “safety cage” architecture is also introduced.

The Handbook is limited to software criticality categories B/C/D software (excluding life critical Cat. A functions).

4.2 Perimeter

In the following subsection, some of the research which have been performed as part of the ad hoc working group is presented. This work does not represent the full extent of research and discussions that were brought about, however, it highlights some of the major findings which will impact the further work on the actual handbook.

Software including the use of machine learning should follow ECSS standards to the extend which is plausible (i.e. tailoring out activities that cannot be performed due to not available artefacts). ML models are very complex, and complete validation / verification of all possible behaviours is typically not feasible. To mitigate the risk from incomplete validation, it is advised to apply SW standards to the maximum possible extend.

However, based on our work so far, it is also clear that the current standards do not cover all challenges coming from Machine Learning, and likewise, that there are aspects of Machine Learning that have to be clearly understood in context of the space industry before we can safely implement such solutions for higher criticality functions.

Based on our findings we now have a high-level overview of ML applications and the challenges related to its verification and validation. Failure mitigation of ML systems is also addressed, initially relying on the concept of a safety-cage architecture.

4.3 Objectives and Challenges

As shown on the Figure 1-1 [EASA Roadmap] AI Taxonomy, section 1.2.1, Machine Learning, and Deep Learning are respectively sub-parts of Artificial intelligence. However, another way of understanding AI, as shown on the Figure 4-1 below, is by differentiate between data driven AI and User-input driven/symbolic AI.

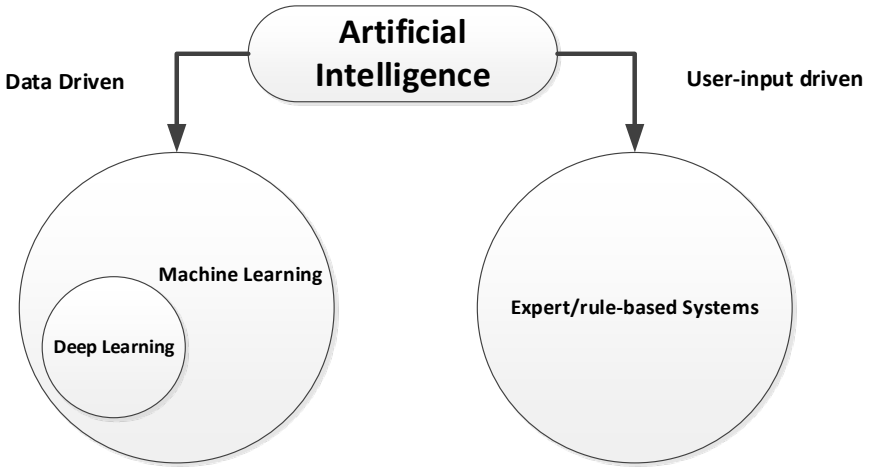

Figure 4-1: AI split into data driven methods

Symbolic AI, which represents artificial intelligence as expert systems, also known as knowledge-based systems or rule-based systems. These systems are created by allowing the user to write "if-then" rules, stored in a so-called knowledge base, that define how a system has to interact with its environment (see [Puppe 1993]). These sort of AI system, have been around since the 1950s, and as their functionality is based around user-written deterministic code, this code can be inspected, reviewed, and tested, following the same practice as presented through software standards ECSS-E-ST-40 and ECSS-Q-ST-80.

However, data driven AI, also known as machine learning, including the sub-field of deep learning, represents “the ability (of AI systems…) to acquire their own knowledge, by extracting patterns from raw data”, [Goodfellow et al. 2016]. Machine learning embodies stochastic processes, e.g. for learning or optimization, as it is trying to teach the models to estimate quantitative values outside of dataset from which the model has been created, i.e. generalization.

However, this stochastic nature also drives a lot of problems for the verification and validation of such data driven AI, which is ultimately what has initiated the work for the ECSS-E-HB-40-02 Handbook.

As per ECSS-Q-ST-80 requirement 5.2.7.2a., under the topic of “Quality requirements and quality models”, a list of characteristics, high-level properties, is presented which shall be used to specify SW quality:

1. functionality;
2. reliability;
3. maintainability;
4. reusability;
5. suitability for safety;

6. security;
7. usability;
8. efficiency;
9. portability;
10. software development effectiveness.

However, when dealing with Machine Learning, many of the above characteristics of a good quality SW model, as per current ECSS standards, become highly challenging to apply, due to the nature of the ML. Not only does the stochastic nature imply issues for such topics as functionality and reliability, but as the model relies mainly on data, the quality of the data and understanding of data representativeness, also has large impact, which currently are questions outside of the above characteristics. And to make things worse, many of the more complex ML models are also black boxes, for which inspection of code does not make sense. A full review of the ECSS-Q-ST-80 standard, including evaluation of the limitation of the above high-level properties was performed by one of the consortium members [ESA-TECQQS-TN-022868].

Part of the inspiration for this review is based on the [DEEL] paper, written by some of the members of this ad-hoc working group, which have derived similar high-level properties, but purely focusing on what good quality ML models should represent, being more considered of machine learning functionality and procedures, and original created outside of space industry setup:

1. Auditability: The extent to which an independent examination of the development and verification process of the system can be performed.
2. Data Quality: The extent to which data are free of defects and possess desired features.
3. Explainability: The extent to which the behaviour of a Machine Learning model can be made understandable to humans.
4. Maintainability: Ability of extending/improving a given system while maintaining its compliance with the unchanged requirements.
5. Resilience: Ability for a system to continue to operate while an error or a fault has occurred.
6. Robustness: (Global) Ability of the system to perform the intended function in the presence of abnormal or unknown inputs / (Local) The extent to which the system provides interchangeable responses for similar inputs.
7. Specifiability: The extent to which the system can be correctly and completely described through a list of requirements.
8. Verifiability: Ability to evaluate an implementation of requirements to determine that they have been met.

As can be seen, many topics overlap with the list from ECSS-Q-ST-80, and as it is written in ESA-TECQQS-TN-022868, minor updates can be done for making this applicable also to ML.

However, as mentioned earlier, due to the stochastic nature of ML, in the training phase, but also sometimes during operation, certain general challenges arise for ML from trying to satisfy such high-level properties [DEEL]:

MAIN CHALLENGE #1: PROBABILISTIC ASSESSMENT

For safety critical systems, quantitative safety analysis is used to assess properties such as "the probability of a catastrophic event of an aircraft shall be lower than 10^{-9} per flight hour". Similarly, Machine Learning techniques rely on mathematic practices that include statistics and probabilities. Nevertheless, despite their similarities, the two domains often employ different definitions and interpretations of key concepts. This makes the endeavour of establishing a safety assessment methodology for ML-based systems difficult.

MAIN CHALLENGE #2: RESILIENCE

Resilience is crucial to ensure safe operation of the system. Resilience typically raises challenges regarding the definition of an "abnormal behaviour", the monitoring of the system at runtime, and the identification of mitigation strategies. With Machine Learning, these challenges are made more complex because of the usually wider range of possible inputs (e.g. images), the difficulties to adopt classical strategies (e.g., redundancy with dissimilarity), and the ML-specific vulnerabilities.

MAIN CHALLENGE #3: SPECIFIABILITY

Specifiability is the ability to describe the intended function of a system in terms of functional, safety, operational, and environmental aspects. This practice allows engineers to designate the target of the development process and to demonstrate that this target has been hit. Nowadays, this is a pillar to build acceptably safe systems. Because ML techniques are often used to address problems that are by nature hard to specify, they raise specific challenges to include them in safe systems, the question of trust being one of these challenges.

MAIN CHALLENGE #4: DATA QUALITY AND REPRESENTATIVENESS

Machine Learning-based systems rely on the exploitation of information contained in datasets. Therefore, the quality of these data, and in particular their representativeness, determines the confidence on the outputs of the ML-based components.

The verification of a dataset with respect to properties related to quality can be particularly complex and depends strongly on the use case.

Conversely, a very representative ML model concentrates relevant knowledge of the behaviour of the physics and behaviour of the systems, which can be reverse engineered from the model.

Therefore, access to the model should be treated with equal care as the raw data themselves.

MAIN CHALLENGE #5: EXPLAINABILITY

The opacity of ML models is seen as a major limitation for their development and deployment, especially for systems delivering high stake decisions. Quite recently, this concern has caught the attention of the research community through XAI (eXplain-able Artificial Intelligence) initiative which aims to make these models explainable. The

ongoing investigations highlight many challenges which are not only technical but also conceptual. The problem is not only to open the black box but to also establish the purpose of explanations, the properties they must fulfil and their inherent limits, in particular in the scope of certification.

MAIN CHALLENGE #6: ROBUSTNESS

Robustness is defined as the ability of the system to perform the intended function in the presence of abnormal or unknown inputs, and to provide equivalent responses within the neighbourhood of an input (local robustness). This property, which is one of the major stakes of certification, is also a very active research domain in Machine Learning. Robustness raises many challenges, from the definition of metrics for assessing robustness or similarity, to out-of-domain detection, and obviously adversarial attacks and defence.

MAIN CHALLENGE #7: VERIFIABILITY

A large and ever-growing set of methods and tools, ranging from formal mathematical methods to massive testing, is available to verify ML-based systems. However, understanding the applicability and limits of each of them, and combining them to reach the verification objectives raises many challenges...

The [DEEL]paper and ESA-TECQQS-TN-022868 both go more into detail regarding how to handle such challenges, e.g., by employing best standards for ML model creation and mathematical proofs, which scope is outside of today's ECSS standards. And it should also be noted that depending on the specific application for ML solution that is being created, the demand to the different high-level properties can be varying. However, as the topic of dealing with these challenges is ongoing research, e.g., explainability, adoption of recommendations on how to handle such challenges should be considered adaptable as the field develops.

Another topic which was explored within the working group has been axes for AI deployment classification, as drivers for the verification and validation process. Within the space domain, there are certain properties driving the verification and validation of software. Examples of such properties are safety criticality, it is the function of the software linked to potential safety critical outcome which could lead to a loss of mission, or on-board versus on-ground applications, i.e. is the application placed on-board a spacecraft with only delayed communication with operators placed on-ground.

The Ad-hoc working group has analysed several potential properties of software application, which might likewise have an impact on the verification and validation of machine learning applications. The attempt with the following list, is to create axes for AI deployment classification that can help the reader to understand the commonalties of the verification and validation processes, between different types of applications.

The list of axes for AI deployment classification is as following:

1. Safety Criticality
2. Complexity of Function
3. On-board vs. On-ground
4. Embeddability
5. Data and Use-case Understanding
6. Online vs. Offline Learning

It can be noted that the first three axes for AI deployment classification are driven by the wished application and the system for which the ML model is supposed to work, whereas the last three axes are driven by the functionality of the ML model itself. An additional quasi-axis, Autonomy, could also be added to the above list. However, since autonomy can be seen as driven by axes 1 and 2 Safety Criticality and Complexity of Function, it was decided to not treat it as its own axis, but more as a special case.

5 Intelligence Environment on ML Verification and Validation

5.1 Objective

The objective of the Chapter is to provide an overview of the initiatives that are relevant to machine learning verification and validation, both within and outside the space domain, at the time of preparation of this Handbook. Initiatives on-going or completed in Agencies and LSIs are primarily collected and within the scope of the Chapter; a brief outline of each activity and its relevance to this Handbook is provided, along with references where additional information can be found.

5.2 Activities / Initiatives in the space domain

5.2.1 European Space Agency (ESA)

The European Space Agency has been actively pursuing the spin-in of AI technologies in the space domain for the last two decades. A number of activities have been completed and some have successfully been operationalised, across different domains of the space mission lifecycle. Detailed descriptions of the vision, activities and domains of interest can be found on the relevant documents [AI@ESA] [Harmo] and the recently completed Artificial Intelligence for Automation (A2I) Roadmap [RD-OpsRm] that covers the application of AI technologies in the spacecraft operations domain.

In collaboration with the European Space Industry, ESA is pursuing the creation of this ECSS Handbook for the verification and validation of Machine Learning applications. In preparation for this task, a Technical Note has been prepared [RD-TN-ESA-Q] to describe the work done to fill the void between Machine Learning projects and Software Product Assurance. Currently, there is a chasm between the two areas, as it has not been possible to standardise the development of ML projects and to ensure process and product quality within a space context; the TN tackles ML development from a product assurance angle.

5.2.2 National Agencies

5.2.2.1 German Aerospace Center (DLR)

The German Space Agency at DLR is funding various machine learning projects across many of its departments like Earth Observation and Navigation. It recognizes generally increasing interest in machine learning, but also in tomorrow's approaches like quantum machine learning that will provide new

perspectives on machine learning and its evaluation. The department of Robotics, Digitalization and Artificial Intelligence has a long history of funding artificial intelligence-related projects with a focus on robotics in space. As a consequence of the increased interest in technologies like machine learning, the group Digitalization, Software and Artificial Intelligence was founded in the department in 2021 to bring forward the respective technologies in the field of space. As one of its topics, the group focuses on supporting the development of technologies for the qualification of machine learning methods for space applications. The group is still under development but will generate relevant results in the next years.

As a persistent funder of numerous projects with machine learning elements, the agency is interested in evaluation and qualification of machine learning systems. It uses the experience from its projects to experiment with indicators and metrics that could help to establish high level data-driven analysis methods. Currently, the set of key figures is being developed and refined. The initial data collection methods are mainly ad-hoc and based on human-in-the-loop survey methods. However, the toolset is actively evolving. The already collected data identifies and compares different evaluation methods. The agency is also preparing to integrate machine learning metrics with the ECSS-based software metrication system AENEAS in order to support monitoring the development of and qualification of such systems. Tried and tested results are expected to be available in three to five years.

The agency is also funding the consortium-based project VeriKI which investigates verification methods for artificial neural networks used in robotics applications. The project sees a major problem of qualifying such systems in the outstanding importance of its training process. It therefore focuses on analysis methods of the trained system to establish evidence of its correctness. While the project has not yet finished, preliminary results have already been provided as input to the standard to the members of the ECSS working group.

By decision of the DLR senate from 2020, a new research Institute for AI Safety and Security has been founded in DLR's research branch. The institute conducts research in the direction of protection against external attacks (security) and reliability of operation (safety). It covers research topics like the engineering of AI systems, AI algorithms, security-critical data, and reliable execution environments for AI. The institute's research will make significant contributions to methods for qualifying machine learning once it is fully established.

5.2.2.2 CNES

CNES considers artificial intelligence as a key issue in its avant-garde policy for the coming years. With the democratisation of access to space, CNES wants to help actors involved with space domain to reach sufficient technological maturity in AI, in order to support the development of digital-related skills. The approach adopted is an investment per project with an exploratory method that will lead to feedback and standardisation once there is sufficient material. Nevertheless, there is crosscutting work in progress on some themes, oriented towards Learning Assurance, such as a study on "model factory": on a similar way CNES

works on legacy software, using such factory for machine learning is studied to empower model production and make it more reliable. CNES is also actively involved in raising the awareness of its teams to the use of machine learning by organizing regular internal training sessions.

5.2.2.3 UKSA

The UK Space Agency has recently published National Space Strategy [RD-UK Space] and National AI strategy [RD-UKAI]; the wide interest in using AI for space is highlighted in both documents. The qualification of AI systems has been found as a very important aspect in adopting the technology operationally, as global technical standards are mentioned to be required for: supporting R&D and innovation, supporting trade, giving UK business more opportunities, delivering on safety, security and trust and supporting conformity assessments and regulatory compliance.

5.2.3 European Space Industry – Large System Integrators

- **Airbus Defence and Space:** As part of Airbus Group, including Airbus Commercial and Airbus Helicopters, an extensive internal AI research roadmap and a dedicated AI synergy project exist= The synergy project called CTAI (Certifiable and Trusted Artificial Intelligence) covers several initiatives around the three directions of “Embeddable AI”, “Certification, Regulation and Standardization of AI”, and “Trusted/Explainable AI”, in addition to an overall project management function to keep an up-to-date holistic view on the different topics within the company, and external developments, to allow maximum synchronization of the different initiatives, while keep exploring potential synergies. Examples of such initiatives which Airbus is part of can be mentioned the [DEEL] paper team, EUROCAE (Section 5.3.2) and likewise this ECSS handbook.
- **Ariane Group:** The Ariane Group has a Working Group dedicated on machine learning qualification. A first issue of company-wide technical operating standard has been internally published to set common grounds for the use of data analytics and artificial intelligence methods in all fields of engineering/manufacturing of space launchers. It is so far mainly dealing with taxonomy and vocabulary. Another internal operating standard is being written with contribution of the whole data science specialist network, dealing with a stepwise process for the construction and certification of machine learning based algorithms. Measures to control data quality are already quite well prescribed. They are a prerequisite for the certification / qualification part, which is yet to be consolidated, although acceptability measures have already been identified.

5.3 Activities / Initiatives outside the space domain

5.3.1 Dependable and Explainable Learning (DEEL) Project

The DEEL (Dependable and Explainable Learning) consortium involves academic and industrial partners in the development of dependable, robust, explainable and certifiable AI technological bricks applied to critical systems; certification experts and AI specialists from the aeronautics, railway and automotive sectors were part of the Consortium. The objective is threefold:

- share knowledge on certification and ML,
- identify the main difficulties raised by the usage of ML in safety critical systems,
- feed the core team with relevant scientific challenges.

After monthly face-to-face meetings, a White Paper [DEEL] has been published covering the full collection of best practices and lessons learned around AI certification, qualification and explainable AI; the White Paper has been used as reference for this Handbook preparation.

5.3.2 EUROCAE / SAE

EUROCAE and SAE are the leading standardisation organizations developing the worldwide recognised industry standards for aviation. Joint EUROCAE/SAE WG-114/G-34 working group "Artificial Intelligence in Aviation" comprised of 500+ members from leading aerospace companies was created in 2019 to develop and maintain Technical Reports on implementation and certification aspects of quickly advancing AI and ML technology for on and off-board aerospace systems and vehicles.

The first document published by this joint working group [RD-EUROCAE] reviews the current aviation assurance practices for safety-critical systems with respect to Artificial Intelligence (AI) / Machine Learning (ML) development approach and provides a list of concerns that need to be addressed in order to produce a future means of compliance for AI/ML-based systems.

The next planned deliverable of the joint working group is a process standard for development and certification of aeronautical safety-related products based on AI/ML-technology. This standard is planned to be published in 2023 and will provide the detailed guidance including objectives and means of compliance for development, verification and validation of AI/ML-based safety-critical airborne systems. This standard is targeting a broad coverage of AI/ML assurance and certification practices and use cases – this is considered highly relevant to the goal of this handbook and can be widely reused and adjusted to space domain.

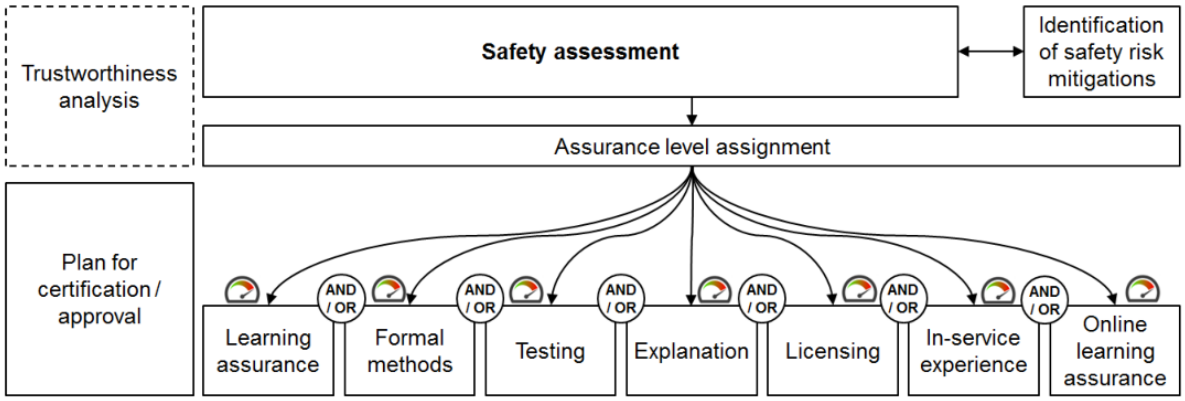

Figure 5-1: [ER-022/AIR6988] AIRBORNE AI/ML ASSURANCE LIFE CYCLE

5.3.3 German Standardisation Roadmap on Artificial Intelligence

In this Standardization Roadmap on Artificial Intelligence [DE-Standard RM], a comprehensive analysis of the current state of and need for international standards and specifications for the technology are presented. In the first edition, the technical, ethical and social aspects of standards in AI are taken into account in detail. The roadmap was drawn up in seven working groups which developed important questions and recommendations for action on Ethics, Quality/conformity assessment/certification and IT security, as horizontal topics, in addition to the basic principles and three AI application fields: industrial automation, mobility/ logistics and medicine. This Roadmap is an ongoing document that is required to be regularly updated in order to reflect the enormously dynamic development of AI technologies and their rapidly expanding fields of application.

A set of additional initiatives can be found on References [RD1- RD6].

6
Guidelines

6.1 Introduction

Machine learning is a rapidly evolving field that has brought a paradigm shift in the way we can solve complex problems. However, the application of ML is not straightforward, and it requires a systematic approach to ensure the quality and reliability of the ML models and systems. In the following sub-chapters, this handbook presents the reader with recommendations of how to approach the verification and validation of machine learning in the usage of space software applications, within criticality categories B/C/D.

In the following sub-sections the topic of verification and validation has been addressed. The structure of the section is as follows. First, the importance of understanding the business value and implications of utilizing machine learning is introduced. Next, a comparison with data-driven approaches and non-data-driven approaches is presented, with a focus on a SWOT analysis. Lastly, specific guidelines are introduced. The guidelines have been split into four sub-sections with the aim of providing a comprehensive and systematic framework for ensuring the quality and reliability of ML models and systems. Each of the three sections focuses on a specific aspect of ML life cycle, from data gathering to system testing, and provides guidance on best practices, testing methodologies, and performance metrics. By following these guidelines, the handbook attempts to provide the reader with an understanding of testing the quality and reliability of ML applications, which is crucial in ensuring the safety and effectiveness of these types of applications.

6.2 Business value consideration

In this section, the understanding of business value from using machine learning is discussed, to provide the reader with guidelines on how to approach the question whether to go for a ML solution or not.

As an initial consideration, organizations who want to optimally utilize data engineering, especially Machine Learning, are often characterized by having the following:

- Strategy for data acquisitions, to assure capturing maximum value.
- Some form of unified data warehouses, to allow teams of developers and engineers to have easy access.
- Some form of development environment, with access to the data warehouse.
- Some form of application deployment environment, allowing streaming of data to applications, and monitoring of performance.

- Dedicated personal to work with the data, e.g. data scientist, data manager, AI/ML product manager, machine learning engineer, etc.

NOTE *There exist a lot of material already on the topic of transforming companies and organizations towards a data/AI approach, this section will not focus on the steps for the transformation itself, but more be looking at some of the technical topics which should be considered as part of the transformation process.*

Second consideration for optimal usage of ML, is the data quality available for model training.

First of all, not all data holds equal value for machine learning and AI. In other words, if the data available does not hold the "correct" correlation to the business objectives, no AI team will be able to create products of value. So, to avoid inflated expectations, it is important to have an overview of available data, and a dedicated team of data/ML engineers to analyse if a given project is possible based on the available input and expected output, before committing to further project developments and results. Hence, the creation of processes of gathering, sorting, evaluating and storing data is important to be able to use it later. This will in turn lead to the questions of data availability/quantity and quality, as further discussed in the following sections.

However, before even getting to these questions, initially there is a certain mind-set which organizations have to adopt:

Just because we have data, does not mean we have to build an AI/ML model.

Just because the problem can be solved with AI/ML, does not mean that it is the best solution.

To expand a bit more on the above statements, the buzz-words of "artificial intelligence", "machine learning" have to be considered in the realistic of usage versus hype. It is true that AI represents a large potential improvement, independent of further discussions on general artificial intelligence, and with the massive amount of data which is being generated on daily basis, it can be assumed that there are plenty of possible applications for machine learning. However, the problem with this sort of mind-set is that it does not consider real business needs: rather than looking at problems and pain-points that often are tied to real value, often organizations and businesses get too caught up in opportunistic thinking. As indicated from the Venn diagram below, Figure: AI versus Value, the aim can from the start be to spend most time working on projects in the intersection, to avoid wasting time and resources. Often seen in the industry, is that projects end up with nice proof-of-concept models, but no further implementation, hence lacking monetary Return-Of-Investment (ROI).

That being said, seeing opportunities based on available data should not be discouraged, however, the difference here is to realistically consider whether the opportunities can be tied to existing business needs before starting, to ensure there exist internal stakeholders, and therefore people who have enough interest in the continued development of the AI project, until a product with value has been created. If not, maybe the resources should be spent elsewhere.

Of course, in cases where the motivation for the usage of ML-based solutions is based on a need from a space mission, the steps are more in reverse, first

exploring what mission requirements are driving a need which ML could solve, realistically evaluating whether non-ML solution could be equally good/better, with less effort, including consideration of data gathering and life-cycle maintenance of the ML solution.

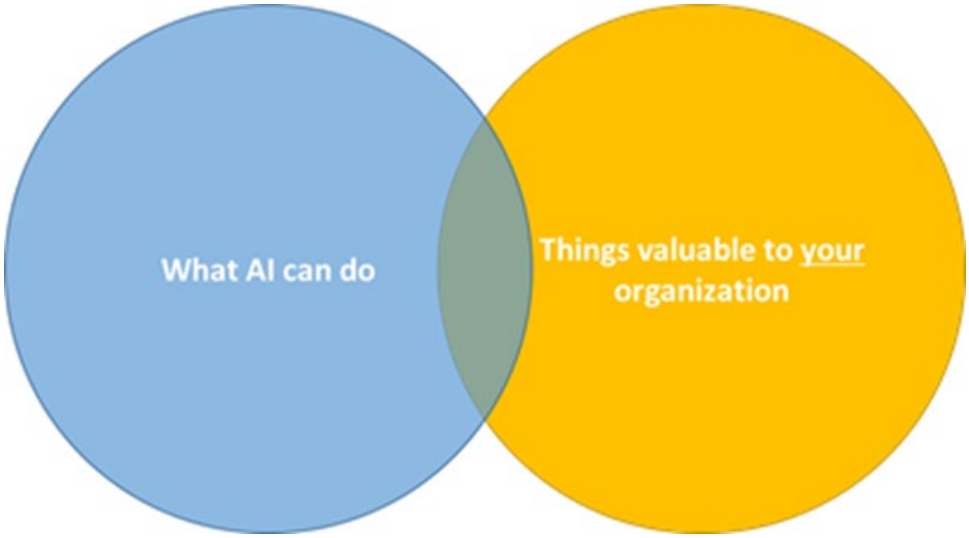

Figure 6-1: AI vs. Value

Once the organization has taken the above statements into account, the next step is to create a good business/usage understanding. This process, specifically for ML, can as a minimum follow these two steps:

- Identify valuable/costly business problems and pain points (which is not the same as specifically ML problems and solutions).
- Evaluate the possibility of using an ML based solution for the problems and pain points, both compared to technical feasibility, business value, and in comparison to non-ML based solutions.

Essentially, ML is supposed to be a part of the general digital workflow and should therefore not be treated separately from normal problem solving, but more as another tool in the toolbox. However, as AI currently still represents a highly specialized skill, it can be noted that finding good AI and ML projects require both AI specialists and also specific domain experts. The domain experts can generally identify issues that need to be solved for the organization ("what are 3 things you wish were working better?"), and then the AI specialists can determine the feasibility of using AI for a possible solution, based on the availability and quality of data, the availability of simulators, or the need for automatization using rule-based systems.

That being said, this does not mean that an AI specialist cannot find some interesting data and come with a suggestion for a project, but the idea can be verified by the domain experts and considered also from a business/product point of view. But the important point is to always consider how a product could be a driver of business value, or how it ties together with pain points for day-to-day work.

For the feasibility of using an ML based solution, technical questions to be answered would be to assess whether it is known how much data would be

needed, if there is a benchmark for the results (from literature or competitors), would it be possible to create an approximation for Bayes error, e.g., compared to human level performance or currently used systems, and what would be the engineering timeline for such a project.

Of business-related questions it is important to clarify the profitability, e.g., quantify what cost can be saved or revenue can be generated by the use of the AI solution. Will the solution substitute parts of existing products/functions, or does it represent something new, and how does it practically affect the workflow (also compared to people's day-to-day work routines)?

Finally, the business and technical teams will have to agree on how to measure performance (evaluation metric). Especially talking about machine learning, the technical team might suggest metrics such as F1 score (e.g. harmonic mean of precision and recall), to show direct performance on the dataset for the ML models to be tried and optimized, but from a business perspective there might also be things that have to be considered, the speed of the algorithm might be linked to revenue generation, or as for anomaly detection for operations, to avoid causing additional work overhead for spacecraft operators we can have minimum false alarms.

A self-explanatory extension of the above-mentioned process, can be seen in Figure 6-2.

Figure 6-2: Simplified process of finding good AI projects

6.3 Data driven approach vs non-data driven approach

In general, the data driven approach refers to a development process based on data and other hard facts, by opposition to historical points, observations subject to various interpretations or even feelings.

In the context of non-data driven approach, a system is classically driven by a set of rules stemming from the knowledge corpus of one or several technical fields. These rules are analytical descriptions of the physical behaviour of the system, stemming from axioms widely accepted by the community. Some of these rules can be tuned to better match real observations or can be adapted to the system of interest, but they form a strong a priori of the dependency structure of the underlying physics.

By opposition, purely data-driven approaches are agnostic of the physical rules: they reconstruct or approximate the physical behaviour suited to the system of interest using only the data from the observations.

Nevertheless, intermediate approaches exist, infusing physical understanding into data-driven models, resulting in hybrid models. For instance, convolution-based approaches are inspired by classical image analysis; recurrent net by higher order dynamical systems.

A data-driven approach can be quite beneficial for a project, however in general it can be avoided when formal models (e.g. mathematical equations) exist already. Below, in Table 6-1, we propose the use of a SWOT matrix as a to quantify the benefits of applying a data-driven approach over a non-data-driven approach.

The matrix can be consulted ***at the start of a project***, in order to conclude whether or not ML is the most suitable solving method for a problem.

Table 6-1: SWOT matrix for data-driven models in ML

STRENGTHS	WEAKNESSES
Satisfactory behaviour in many cases when confronted with unseen input conditions (2) Enabling new applications Performance, often close or beyond human standards Industrial property not easy to steal (1) Moderate computing requirements at inference Quick development of a proof-of-concept	Stochastic behaviour, lack of confidence intervals Data dependant (in particular, sensitive to corrupted or biased data) High computing power required to train and validate the model from the data; accessing a huge quantity of data can add strong infrastructure constraints Limited explainability for a single inference Subcontracting the design of the model requires to share the entire dataset
OPPORTUNITIES	**THREATS**
Business reusability across different applications (2) Large community of enthusiasts of data-driven models Many freely available repositories of published or work-in-progress models, available for reuse in space applications (3) Rising trend in hybrid solutions, mixing data- and non-data-driven models Availability of specialized hardware for data-driven model generation and evaluation: performance boost with affordable investment	Speed of evolution of dedicated AI libraries and HW leading to maintenance cost Reduction of the key players in AI libraries leading to monopoly or high dependency Overconfidence in the capacities of fully data-driven approaches in the literature Underestimation of the industrialization and maintenance of data-driven models

(1) since a ML model consists of a model architecture, a set of hyper-parameters and large parameter set, it is not easy to publish and quite inconvenient to steal. Partial information about the model is almost useless.

(2) unseen input is still required to come from the same distribution as the training data to avoid out-of-distribution errors

(3) such published models can be reused in their entirety, or just partially (transfer learning) or selectively retrained for a specific application.

6.4 Guidelines

6.4.1 Data Quality

Data quality is of primordial importance in ML based applications. Low quality data can lead to inaccurate output and an improvement in the quality of the data can enhance dramatically the efficiency of the application. It is often seen [e.g. Jain et al. 2020] that improving the quality of the data leads to greater improvement of the global application outputs than the fine tunings of the model used.

This section starts by describing high level properties that help characterizing the quality of the data. Then the data lifecycle of a typical ML project is described, as well as the concept of operational scenarios. As data can originate from various sources, these sources and their impact on data quality is discussed. Finally, guidelines related to specific ML applications are given.

It can be noted that the definition of data includes all meta-data attached to it. In case of supervised learning the quality and the consistency of the labelling is of the greatest importance. For example, in time series, the time stamps of the data are essential, whereas for images the knowledge of camera characteristics can be of great help.

The type, amount and completeness of relevant metadata are as important for the quality assessment of the data as the quality of the actual data. In the following sections the term “data” includes all relevant meta-data that could be attached.

It can also be noted that the **data quality assessment process** described in this chapter is **not a one time job**, but it is advised to do continuously through the project as there are often iteration loops in the development process.

In order to formalise the concept of quality for data, several high-level properties are introduced in the next section, which describes in more detail how these properties can be evaluated for specific cases. Several guidelines for data manipulations are then also proposed.

6.4.1.1 High level properties

The DEEL white paper [DEEL] defines the following properties for data quality:

- **Accuracy** depends on data gathering/generation and measures the faithfulness to the real value. It also measures the degree of ambiguity of the representation of the information.
- **Accessibility** measures the effort required to access data.
- **Consistency** measures the deviation of values, domains, and formats between the original dataset and a pre-processed dataset.
- **Relevance and Fitness**, with two-level requirements:
  - The amount of accessed data used and if they are sufficient to realise the intended function and
  - the degree to which the data produced matches users’ needs.

- **Timeliness** measures the "time delay from data generation and acquisition to utilization". If required data cannot be collected in real time or if the data need to be accessible over a very long time and are not regularly updated, then information can be outdated or invalid.
- **Traceability** reflects how much both the data source and the data pipeline are available. Activities to identify all the data pipeline components have to be considered in order to guarantee such quality.
- **Usability** is a quality bound to the credibility of data, i.e. if their correctness is regularly evaluated, and if data exist in the range of known or acceptable values.

These properties are very generic, and it is likely difficult to directly try to assess quality from them in a specific project. However, we might try to specify them a bit further. Accessibility here refers to the effort required to obtain the data. It can be the physical retrieval which can be difficult because the data are stored in a satellite for example or the legal access to the data can be difficult because they are proprietary. For the Relevance and Fitness, whether or not the data match the user needs refers to content of the data and the specific application at hand. For example, perfect image of clouds could check all other properties but are not relevant for searching for boats in ocean or assessing the state of forests. Whether or not the data are complete or not, whether they are a lot of missing values or not, this would then fall into the Usability property. Considering Timeliness, it is advised to verify obsolescence criteria if any.

In addition, the concept of quality for the data is strongly application dependent. A good quality set does not have the same meaning for the application of a supervised learning on images and for an unsupervised learning on time series for example.

The documents [ESA-TECQQS-TN-022868] and [EASA-Concept] go further than the high-level properties and propose specific verification points that can be applied to all data regardless of the applications. Verifications that can be considered at start before going further in the data quality assessment are:

- Legal/ethical aspects are considered
- The format of the data is suitable for the work at hand
- Possible missing or duplicated values are addressed
- Possible bias or noise in data are searched for and addressed.
- The possible need for data augmentation is analysed
- The origin of the data is known as well as the pipeline of operations applied on them (traceability)
- A mechanism is defined to ensure data will not be corrupted during storage and processing.

In order to further refine verifications of the data quality, the discussion can be targeted on more specific applications as it is done in the following sections.

6.4.1.2 Data lifecycle

6.4.1.2.1 Overview

At the beginning of a project involving ML, a first assessment about data quality is recommended to be performed. As part of this assessment, it is recommended to perform an exploratory data analysis (EDA) not only to assess the quality properties defined before but also to understand the underlying structure of the data. This includes the identification of patterns and trends, anomaly and outlier detection, and an evaluation of the data's fitness with respect to the requirements associated with the ML model to be developed. As the types of data and the available amount can rule out certain types of ML models, this process is advised to be performed **first in the project timeline**, and in any case before any model selection. The document [ESA-TECQQS-TN-022868] suggests a possible milestone that concludes this review process called Data Readiness Review. In the document, a set of verification and data properties to consider are listed as a methodology to assess the quality of the data, such as: accessibility of data, faithfulness and representation of data, and data context.

The development of a component using ML is usually an iterative process. Once data are gathered and a first assessment of data quality is performed, the next step is to prepare the data for training and testing. The general guideline for data splitting is to create three different datasets: training datasets, validation dataset and test dataset. The first two, training and validation datasets, will be used during the learning (training) phase, which involves hyper-parameter tuning and model's weights fit. The last one, test dataset, should be used only once, to evaluate the performance of the best candidate model. The final size of each dataset can be determined per use case, in accordance with different factors such as total data size available. As a reference a common percentage split goes in the line of 80-10-10/ 70-20-10 for training-validation-test, usually in the cases where less than 10.000 data samples are available for the training. However, when using substantially large datasets, above 1 million data samples, splits such as 98-1-1 starts to be appropriate. Data splitting is one of the most sensitive steps in the data lifecycle as it will impact both: the learning and evaluation phase. The following are aspects to take into consideration when performing the data splitting.

- Dataset representativeness: All the possible cases, that the model needs to consider, should be properly represented in the data distribution of each of the datasets. Note that missing scenarios from the training/validation datasets would lead to an underperforming model . In such case, the model could underperform to the point of providing wrong or inaccurate answers which would question the model robustness or resilience. Missing scenarios from the test dataset would lead to an inadequate evaluation of the model performance. The concept of "operational scenarios" is introduced later to help with the representativeness assessment of the data.
- Hold-out test dataset: The test dataset can be kept unmodified during the model development process and effort should be put to ensure there is no data leakage from the training/validation datasets. This also means that new data generated with techniques like data augmentation should only be used to increase the representativeness and size of the training and

validation datasets but never be included in the test dataset. If the data sets nevertheless need to be updated because of mission definition consolidation, it is advised to re-assess its quality characteristics.

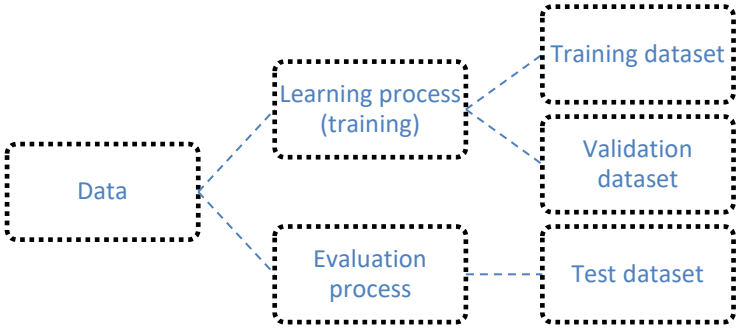

Figure 6-3: Data Process

The training-validation-test splitting is the preferred approach, however there are cases when it can be difficult to apply, especially when working with small datasets. In these cases, a data split considering only two datasets namely training, for the learning phase, and test, for the evaluation phase, can also be considered. However, such a split is only recommended if bias is not an important factor for the model. Alternatively, k-fold cross validation procedure can also be applied, especially when the dataset is too small to ensure enough representativeness through the data splits.

If during this process, bias or data quality related problems are discovered, the data set could be updated to address the issues and the data split process can be initiated again afterward. In many cases, the original data set goes through a set of transformations to be more suited to the model training (ex. categorical data treatment). At each evolution of the data set, it is advised to repeat the quality assessment.

It is also advised to store all data splits used during the training, validation and testing process to ensure reproducibility of the entire process.

After the system quality has been assessed and it has been deployed, data used by the system in its business application can be monitored on a regular basis, so that any drift between the test set input distribution and the real application input distribution can be detected and investigated. Even if input values remain within the span of test set values, data drift can negatively impact the system performance by operating the ML model more frequently in input subsets with lower performance.

6.4.1.2.2 Operational scenarios and operational design domain

To assess the representativeness of the data, one can consider the concept of operational scenarios and operational design domain (ODD) to ensure that all cases that will be met in real life are met in the data.

The definitions of operational scenario and operational design domain are taken from [EASA ConceptPaper]. Operational scenarios are defined through the definition of the associated operational designed domain, that describes the operating limitations and conditions associated to the proposed operations for the AI-based system. The ODD, for instance, can define the value range for the different operating parameters or enumerate the valid values for categorical parameters. Therefore, the representativeness of the data can be assessed by comparing the data population against the defined ODD.

Figure 6-4 from [EASA ConceptPaper] depicts the relation between operational scenarios and operational design domain.

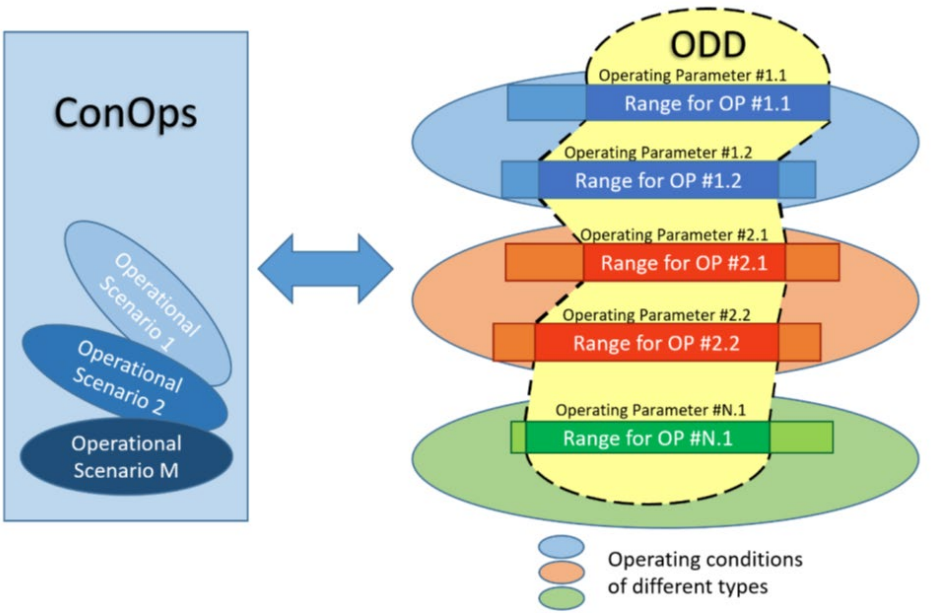

Figure 6-4: Operational Design

For example, in the case of a data set made of images, it is advised to ensure that the data set contains pictures with the range of variations, defined by the ODD that will be met by the applications, in particular addressing worst-case scenarios and corner cases. For instance, luminosity variations or background variations.

In the case of time dependent data like time series, the time tag of these data is of course of primordial importance. These data are usually periodic coming from a sampling at a given rate or a telemetry generation at a given frequency. It is advised to search for missing data, holes in the time series and to identify a method to deal with the issue. Depending on the specific application, a certain precision in the time tag can be required.

It is thus advised to define operational scenarios and the associated ODD to verify the relevance and fitness of the data and to ensure that the model is trained, validated and tested on a data population that is representative of the real data population.

In addition, the relevance analysis can be supported by matching the data sources with the system engineering models that can be provided using the MBSE methodologies and tools.

The concept of operational scenarios and ODD is also relevant to ensure the correctness of the model interfaces or to set the context for the model testing (see 6.4.3 “Machine Learning Model Testing”).

6.4.1.3 Data sources

6.4.1.3.1 Overview

Before entering the various possible applications, the data sources are discussed. The types of data identified are:

- real data
- simulation/synthetic data
- augmentation data
- surrogate data
- lab data.

Although all properties defined in the previous sections for data can be applied to the various types of data, their importance vary. For example, representativeness is not so much an issue for real data, but it is for the other types. On the other hand, completeness and labelling are typical challenges for real data but are rarely a problem for the other types.

6.4.1.3.2 Real data

Data can be gathered by sensors, camera or recorders of any kind, in this case, these are real data. The representativeness of this type of data is not questioned but the fitness of these data for the application at hand can be questioned; for example, pictures of clouds are of no use in the training of a model trying to recognise boats in satellite images.

Real data comes with important metadata such as the date, the type of sensors, etc. These metadata can be used to assess quality through the high-level properties listed above. In particular, the accuracy and usability by verifying if the data were gathered by a properly functioning sensor and if the data received were recorded in the expected range of data acquisition for this sensor. For timeliness, the presence of a time tag is essential.

6.4.1.3.3 Simulation and Synthetic data

Simulated data are generated by a highly representative complex simulation or a Digital Twin. For example, the telemetry generated by an on-board software running on an operational simulator. Such simulators run the actual Flight Software usually on a processor emulator and simulate all satellite equipment's as well as the space environment (Sun and Moon positions, forces and torques applied on the satellite etc...). When using such high-fidelity simulators, differences between these data and real data are generally very limited, albeit hard to quantify. They still could cause bias in the training process; it is thus important to assess their representativeness.

Other data, not specifically generated by a dedicated simulation, called synthetic data, can also be used. For example, it can be images generated by a 3D modelling tool that allows to finely reproduce the expected image that will be seen from a

camera. It could also be data generated by a functional model of a given sensor. The difficulty is assessing their representativeness is the same as for simulation data.

For this type of data, assessing the accuracy is the key. There are rarely missing data problems, and more data can always be generated. If measurement data is available, the measurement data can be used to calibrate the simulation model.

In space application, real data are rarely accessible and when they are, there is often missing data. This means that in practice simulation and synthetic data will often be used.

6.4.1.3.4 Augmentation data

Synthetic data that are based on real data and used to augment the data set are another type of data. They are closer to real data and have less representativeness problem. However, they can still introduce bias in the ML training and should be generated with caution.

The accuracy and consistency are the properties on which to focus for this type of data.

6.4.1.3.5 Surrogate data

In the case where data are proprietary and cannot be used directly or in order to ensure confidentiality, it is good practice to anonymise data or modify in a way that does not modify the underlying information we want to learn from the data. In such case, data are called surrogate.

6.4.1.3.6 Lab data

Lab data are somewhat in between real data and simulated data, they are data generated using the real equipment but, in a lab and not in space. The main problem with this type of data is their amount as they are time consuming to generate. Furthermore, their usability and relevance and fitness need to be assessed. Although they are generated by real equipment, they are not in the space environment. This implies that some effects like change in temperature or luminosity can induce a lack of representativeness on the generated data.

In practice, real data are rarely available, either because the mission is not yet launched or because the data are sensitive and not shared. It is thus often required to mix these various types of data to develop the intended application

6.4.1.4 Specific applications

6.4.1.4.1 Supervised learning

Supervised learning represents the training of machine learning models using labelled data, providing specific examples of what the model should learn. In the case of supervised learning, the quality of the labelling is essential. It is advised to have a close involvement with experts of the field to make sure to build an understanding of the data at hand. Giving precise rules for labelling can also help reaching an acceptable consistency in the labelling. Standardization of the labelling procedure is key, especially whenever labelling is performed by humans.

Especially in the case of citizen science (i.e. crowd labelling) it is advised to check the labelling in the assessment of the quality of the data.

6.4.1.4.2 Unsupervised Learning

In the case of unsupervised learning, labelling is not relevant. For example, this type of technique can be used to learn a generic behaviours (e.g. Nominal behaviour) such that later any different behaviours (e.g. Faulty behaviour) can be identified. In that case, the data set is expected to represent the targeted behaviour in all its variations otherwise, a misclassification can occur. This example falls into the search for bias in the data.

6.4.1.4.3 Reinforcement learning

Reinforcement learning is a type of machine learning that involves training an agent to make decisions in an environment to maximize a reward signal. The agent learns through trial and error, by taking actions in the environment and receiving feedback in the form of rewards or penalties. The goal of reinforcement learning is to develop an optimal model, which contain is a set of rules that the agent uses to select actions based on the current state of the environment, to maximize a reward scheme.

In reinforcement learning (RL), there are two primary learning approaches: model-based and model-free learning, [neptune.ai]. Both approaches can utilize environments built from real data or simulations, but they differ in how they use this information.

Model-based learning involves constructing a detailed model of the environment, which is then used to simulate future states and rewards. This allows the agent to engage in extensive planning and predictive decision-making. The model provides the flexibility to explore different scenarios without the need for actual interactions, enhancing sample efficiency. However, the success of this method is contingent upon the fidelity of the model to the real environment, and it requires accurate and representative environmental data or simulations.

Model-free learning, on the other hand, does not involve building a model of the environment. Instead, it learns a policy directly through trial-and-error based on real or simulated experiences, relying entirely on observed state transitions and rewards. This method is generally simpler and more robust, as it is not susceptible to inaccuracies in a model. However, it usually requires a larger volume of interactions to develop an optimal policy due to its direct reliance on data for learning.

Both simulated and real data environments have their applications in either approach. Simulated environments are particularly valuable in situations where real interactions are too costly, risky, or impractical. They allow both model-based and model-free methods to operate under controlled, repeatable conditions for training. Real data environments offer the advantage of training and testing agents in real-world conditions, providing them with realistic challenges and variabilities that are difficult to simulate accurately.

Choosing between model-based and model-free learning, and deciding whether to use simulated or real data, depends on several factors, including the specific goals of the application, the availability and quality of data, the computational resources at hand, and the acceptable trade-offs between accuracy, efficiency, and robustness. For the topic of verification and validation of data vs simulation, please see section 6.4.1.2.

6.4.1.4.4 On Board training

On Board training comes with additional challenges regarding data quality. For example, in the case of supervised learning how to ensure proper labelling of data? In the case of unsupervised learning, how to ensure the faithfulness of the data (for example data from a broken sensor could introduce a bias)?

Since on board training is unlikely to be used in space application in the close future, it is left outside the scope of this document.

Possible solutions could be to compute data health checks on board or download samples of data used for re-training to the ground on a regular basis and to perform a data quality review on these data. In such case, the guidelines listed above apply directly.

6.4.2 Model development process

6.4.2.1 Overview

Once the data set is selected, and its quality has been assessed, the ML model to train is chosen. This section describes the best practice to select a model. After a brief discussion on frameworks, a list of model characteristics is given. Then model selection itself is discussed. Finally, common issues that can be encountered when working with ML and a possible solution to overcome them are given.

The overall workflow in the model development process is the following:

- As mentioned already in the section “Data lifecycle” the data is split into different datasets.
- The training process begins, in which the model is fit. The training process will also involve the tuning of the different training hyper-parameters, such as: optimizer algorithms, learning rate, or epochs. During this phase different models can be trained in parallel, helping to find the best candidate (see 6.4.2.4 Model Selection).
- Once the performance of a model in the training phase is found adequate the next step would be to evaluate it against the test dataset. The evaluation of the model against the hold-out dataset will give an indication of its generalization capabilities.

After either the training or the validation part, it is possible that problems with the data or the selected model are encountered. In case of a data issue, it is advised to update or correct the data set and re-assess its quality. Then the model selection process can be repeated. Alternatively, multiple iterations on the model selection or design (optimisation) might be needed to meet specific performance (e.g. accuracy) or inference requirements (e.g. speed, power, etc.).

It is important to mention that the development process of most machine learning algorithms is a stochastic process. For instance, stochastics optimization algorithms such as stochastic gradient descent are used to find the optima during the training process. For this reason, the outcome of the training process can be different each time, even if the same training process is followed using the same data.

It can also be mentioned that despite that the aim is to create the best possible model, there is a theoretical upper limit of performance that the model can achieve, also known as Bayes Optimal Error, see Figure 6-5 as it is practical impossible to build a perfect system (containing all information).

However, through reduction of uncertainty of the model or data, e.g., based on error-analysis or adding use-case-specific knowledge, it is possible to get closer to the Bayes Optimal, which is an important part of training ML models if high performance is demanded (related to performance need vs. effort to achieve such performance), but is especially important for ML models if intended to be used for safety hazardous systems. Hence, a lot of ongoing research efforts are going towards such reduction: from a critical consideration assessment of the data quality (See section 6.4.1 "Data Quality") to the application of specific tests to reduce model uncertainty (See section 6.4.3 "Machine Learning Model Testing").

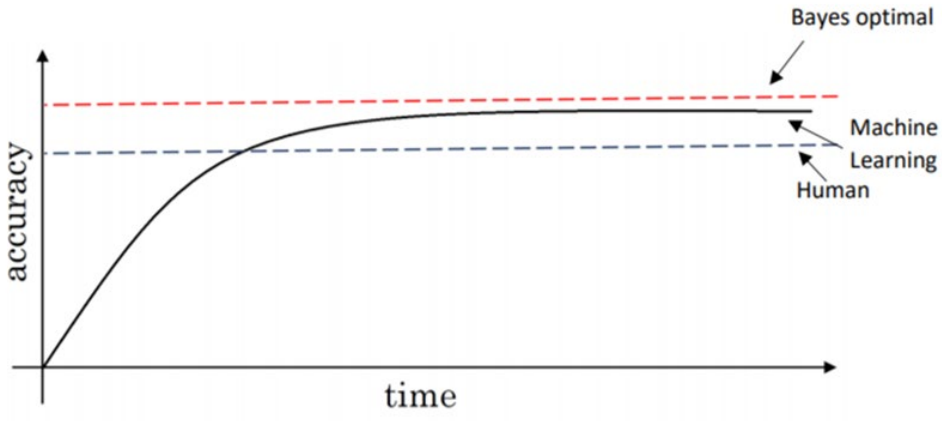

Figure 6-5: [Ng et al.] Example of theoretical upper limit (Bayes Optimal Error)

Of course, the Bayes Optimal Error is more of a theoretical concept, however, by defining a proxy for this error it is possible to use Bayes optimal error as a benchmark to evaluate the performance of a ML model and identify areas for improvement. Examples of such proxies could be a standard software solution or a group of experts, which you can test on the same tasks as the ML model and compare performance. This is especially interesting within deep learning where with the right amount of quality data and models for some use-cases the performance can get better than human abilities, for example image recognition. By comparing the model's performance to the proxy, it can help on to approach further minimization of bias and variance, beyond what can be seen from training on an isolated dataset, and provide a more accurate understanding of how the ML model is likely to perform in the real world.

To use an approximate Bayes optimal error the following steps are recommended:

- Train the model following the guidelines already defined in previous sections (See section 6.4.2 "Model development process").
- Once the ML model has been trained, find a proxy which can be used as the approximate Bayes optimal error, to be functioning as benchmark for performance evaluate of the model. This could involve consulting with

domain experts to measure their performance of the task, or using other software or models that are known to perform well on similar tasks.

- Compare the performance of the trained ML model to that of the approximate Bayes optimal error based on the test dataset. If the model's performance is significantly worse than the Bayes optimal error, this could indicate that there are underlying issues with the model that need to be addressed. On the other hand, if the model's performance is close to the Bayes optimal error, this suggests that the model is performing well and can be ready for deployment.
- Iterate and refine the ML model as necessary. If the model is not performing as well as expected, it might need more good quality data, or the model type might not be good enough in capturing the complexity of the data and it would be worth trying a different type of model altogether.

6.4.2.2 Framework

A wide list of frameworks for the development of ML applications are available like TensorFlow, PyTorch, Scikit-learn, etc. No framework is identified as preferred for the development process and choosing one over other answer to different factors such as: previous experience, functionality provided or compatibility. However, in the case of cat C or B, it is advised to consider the qualification need of the inference engine required to execute the model or to consider compensating measures.

6.4.2.3 Model quality characteristics

The following characteristics are associated with machine learning model quality. These definitions are taken from [ESA-TECQQS-TN-022868] which adapts the concepts from general software quality characteristics to machine learning).

- **Functionality**: The capability of the ML model to provide functions which meet stated and implied needs when it is being used under specified conditions.
- **Reliability**: The capability of an ML-based component to maintain a specified level of performance when used under specified conditions.
- **Robustness**: Robustness has two different definitions:
  - **Local robustness**: The extent to which the system provides equivalent responses for similar inputs.
  - **Global robustness**: Ability of the ML component to perform the intended function in the presence of abnormal or unknown inputs.
- **Resilience**: The ability for a system to continue to operate while an error or a fault has occurred.
- **Explainability**: The ease with which a human can comprehend an ML model, its data, and its results and outputs. This characteristic is especially noteworthy for the validation, debugging, and program approval of ML models, as well as for any system that involve a Human in the Loop (for details and delineation from interpretability see 6.4.2.4.2).

These quality characteristics help to build a proper quality reference framework to support different decisions during the machine learning development life cycle such as the model selection and/or model testing phases. It is important to mention that some of these characteristics can already be initially evaluated during the model evaluation phase while other need to be assessed during the model testing phase (see section 6.4.3 "Machine Learning Model Testing").

Table 6-2 describes to which phase these characteristics can be initially associated. This distinction assumes that in the model selection only the performance metrics associated with the test dataset are evaluated while in the model testing other techniques are applied.

Table 6-2: Phases and characteristics

Characteristics	Model selection	Model testing	Rationale
Functionality	✓	✓	Functionality can initially be evaluated during model selection using the performance evaluation metric of the model on the test data set.
Reliability		✓	Reliability is better evaluated during the test phase. See section 6.4.3 "Machine Learning Model Testing".
Robustness	✓	✓	Robustness can be initially evaluated by augmenting the training dataset with noise. Other techniques like OOD testing or adversarial testing can be applied later.
Resilience:		✓	Resilience can be better tested on an already trained model.
Interpretability/ Explainability	✓	✓	Model interpretability can already be assessed at model selection as it is an intrinsic characteristic of each model. Some XAI method can also be evaluated during model selection as they can be included in the training phase, like attention mechanisms for instance. Other XAI method can be better evaluated during the testing phase.

Finally, it is important to understand the concept of stochasticity at model level. In the previous section the stochastic aspect of the training process was introduced, in this sense, machine learning applications are also usually referred as being stochastic. The reason is twofold, firstly some machine learning model, such as probabilistic models, are inherently stochastics. Secondly, there are multiple sources of uncertainty associated with machine learning applications, such as aleatoric or epistemic uncertainty. Note that, most machine learning models, once they are trained and their weight are fixed are not stochastic and have a deterministic behaviour, always giving the same output for the same inputs. However, even in this case, they are generally referred as being stochastics because of the level of uncertainty associated with the output when new inputs are processed.

6.4.2.4 Model selection

6.4.2.4.1 Overview

Given the ease with which machine learning models can be applied using popular frameworks it is important to distinguish between experimentation for the sake of learning and model training in projects. In the latter case it is advisable to refer back to the section 6.2 "Business value consideration" in order to test whether the problem to be solved is adequately defined, including an integration of model metrics with the expected business impact. A clear understanding of the project's goal will facilitate with model selection: requirements for the models can include, aside from technical aspects such as required classification thresholds or challenges related to the data, a consideration of relevant constraints such as

- Need for explainability (See section 6.4.2.4.2 "Machine Learning Interpretability and Explainability")
- Availability of resources for training and at inference time to meet the target performance (speed, latency, etc.).
- Sensitivity of data with respect to data privacy and confidentiality)
- Amount of data available
- Need for formal mathematical validation
- Anticipate and mitigate possible findings from post-training optimization

Awareness of constraints will support a better definition of the solution space and likely remove potential model families from further consideration. Good practices regarding project definition prior to model fitting are available in the literature on design of experiments.

Generically, it is advised to define requirements that need to be fulfilled by the model. The model selection can then be done following these requirements. It is expected for the process to be iterative as it is likely that several models are selected at first. Following several tests and performance evaluation, the list of possible models then shrinks towards the best one for the application at hand.

It is advised to always start by testing the simplest model that can satisfy the needs considering the available data quality and quantity. This initial exploratory analysis can also include, when possible, given the requirements, non-ML-solution. The attempt of going directly to a more complex model can add complexity to the solution for the wrong reasons. It is in the case where simple models cannot learn the underlying pattern or show large bias in their predictions that the use of more complex models is justified. For deep learning models it is advised to, if possible, start with an architecture found in the literature for which often implementations are available.

However, as the model complexity, the chances of a model showing large variance in the predictions or signs of overfitting the training data increase. For this reason, it is advised to always consider the bias and the variance of a given model and to minimise them as much as possible. The trade-off between variance and bias is a key aspect to consider in the model selection as high variance is associated with overfitting while high bias is associated with underfitting We

mean by variance in this context, the variation of a given ML model when assessed on different data sub-sets. Data bias is defined further down in this document.

Depending on the application and its criticality, it might be useful to consider in the selection process the possibility to formally compute boundaries on the performance of the model. Such proof can be obtained through statistical learning theory (SLT) for example. However, such mathematical boundaries on the performance of an ML model cannot be computed for all types of models.

It is advised to favour explainability in the model selection when possible. In case of errors, this will allow a better understanding of the problem. However, sometimes an application does not gain from explainability, and performances could be favoured.

In order to compare the performance of a model, it is advised to use benchmark datasets whether public or internal datasets, whose adequacy is confirmed by specialists of ML and technical domains. If the dataset is generic and not related to the application at hand, such performance evaluation can be limited to first-order trade-off. This allows to rapidly compare performances of various types of models. However, a careful selection of the benchmark is important as the benchmark data are expected to represent as much as possible the complexity of the data at hand.

An important habit is to keep track of all steps leading to model selection such as the various model tested, the framework (its exact version), the data set etc. It is important to justify the model choice and to be able to reproduce the selection process.

6.4.2.4.2 Machine Learning Interpretability and Explainability

In previous sections the need for explainability was identified as a key driver in the model selection process. Explainability, and the closely related concept of interpretability, are fundamental concepts in machine learning and therefore it is important to understand their meaning and impact in the machine learning life cycle.

Explainable Machine Learning is a set of methods/tools that help to understand predictions made by the ML model. In the field of machine learning the term explainability is commonly used interchangeably with “interpretability” however, although related, they have different meanings. In the context of the handbook the meaning of “interpretability” is borrowed from [RD Miller2017] and is defined as “the degree to which a human can understand the cause of a decision", It is thus a property of the algorithm. In contrast, explainability, refers to a set of mechanisms aimed at making algorithms that have intrinsically low interpretability more understandable. T Explainability methods are especially useful when applied to algorithms with low degree of interpretability. From a practical point of view, there are algorithms that have a better degree of interpretability, like linear regression or decisions trees, than others, like NNs: is for the latter group of algorithms where explainability methods are most valuable.

The need for interpretable or explainable ML models becomes more obvious in higher-risk environments, where a mistake could result in serious consequences;

for certain problems or tasks it is not enough to get the prediction (the what), but the model must also explain how it came to the prediction (the why), because a correct prediction only partially solves the original problem. Other reasons why interpretability and explainability are important for ML applications are found in detecting bias and ensuring fairness, increase social acceptance/trustworthiness, debugging and auditing, ensuring reliability or robustness and checking causality.

At the same time, ML gets a bad reputation when it negatively impacts business profits. This often happens because of disconnection between technical team and the business team. Explainability has the potential to connect the technical people and non-technical people, improving knowledge exchange, and giving all stakeholders a better understanding of product requirements and limitations.

6.4.2.4.2.1. Taxonomy

Different taxonomies can be found for machine learning interpretability and explainability methods, and it is a very active field of research, which means the taxonomy presented here should not be seen as exhaustive. In Figure 6-6, **Figure 6-7** and Figure 6-8 one high level classification is depicted starting with intrinsically interpretable models and moving to explainability methods to apply to less interpretable algorithms. This taxonomy is taken from [INT-ML] and for more information on the specific methods it is advice to go to the original source of the reference.

Intrinsically interpretable *models* **Refers to ML models algorithms that are considered interpretable due to their simple structure.**
Linear regression
Logistic regression
Decision trees
RuleFit
Naïve Bayes
k-nearest neighbours

Figure 6-6: Examples of classical ML techniques

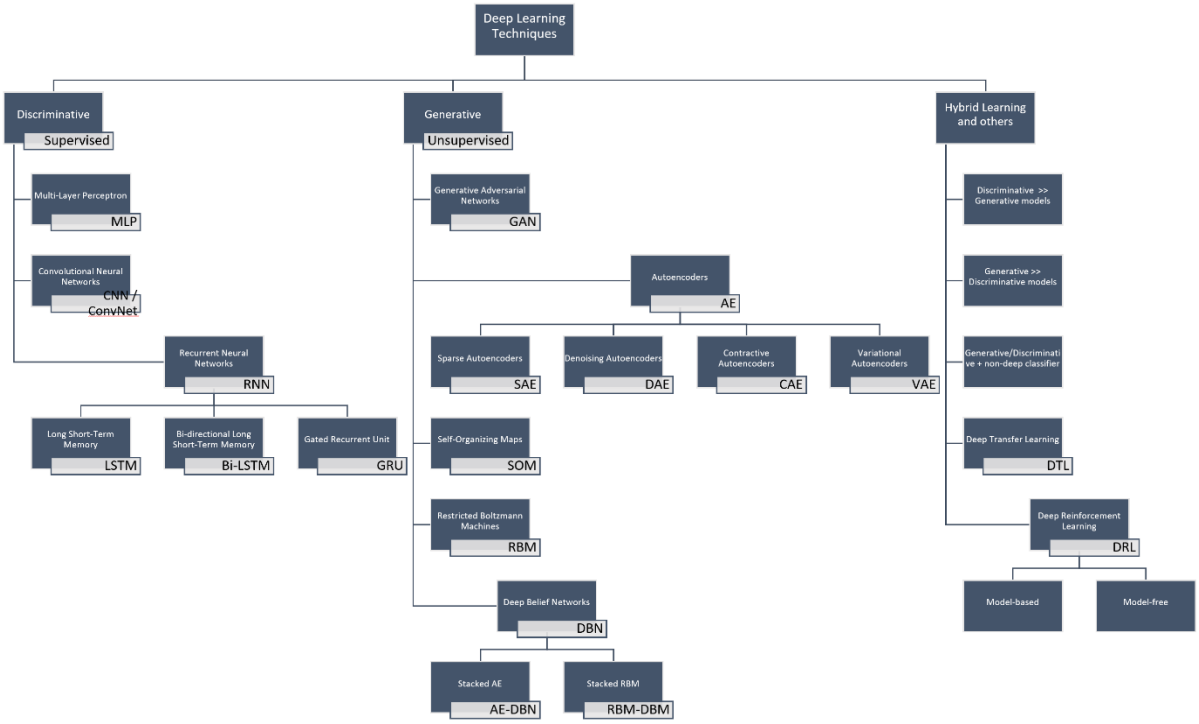

Figure 6-7: Taxonomy of deep learning techniques from [Sarker 2021]

Post-hoc interpretation *methods* **Refers to the application of explainability methods after model training.** ***Note*: They can also be applied to intrinsically interpretable models.**		
Global Model-Agnostic methods **Describe the average behaviour of an ML model**	**Local Model-Agnostic** Explain individual predictions	**Neural Network Interpretation**
Partial dependence plot	Individual conditional expectation curves	Learned Features: - Feature Visualisation - Network Dissection
Accumulated local effect plots	Local surrogate models (LIME)	Pixel Attribution (Saliency Maps)
Feature interaction (H-statistic)	Scoped rules (anchors)	Concepts
Functional decomposition	Counterfactual explanations	Counterfactual explanations
Permutation feature importance	Shapley values	Influential instances
Global surrogate models(*)	SHAP	
(*) **If a coarser global model still makes sense for expert review.**		

Figure 6-8: Post-hoc interpretation methods

Figure 6-6, **Figure 6-7** and Figure 6-8 describe methods to be applied post-hoc that is, once the model has been trained. There are also other methods that can be applied during training, like attention mechanism.

As a practical guideline one can start, whenever possible given the specific application, with algorithms that are less complex but with a higher degree of interpretability and then move to more complex algorithms when the performance is deemed not sufficient, understanding that more complex algorithms can achieve better performance but at the cost of interpretability. This trade-off between complexity, interpretability and performance needs to be done carefully, per application, avoiding the use of more complicated algorithms when the delta increase in performance does not justify the increase in complexity and the decrease in interpretability.

It is also important to understand that the different explainability methods provide only a partial understanding of the algorithm's behaviour and are subject to its own limitations, therefore, any conclusion derived from its application needs to be taken carefully. To give some context in [STOP-BB] some general limitations and problems associated to different explainability methods are described.

6.4.2.5 Common issues with Machine Learning model training

6.4.2.5.1 Overview

This section aims to highlight common issues encountered in the training of ML models and provides references for further information on how to avoid these pitfalls. An awareness of common issues is deemed important due to the wide-spread popularity and ease of use of modern machine learning framework. Conversely, the section does not aim to be exhaustive and should not be regarded as complete list of all potential errors that can be introduced into model training. For additional information please refer to [Lones 2021], [Ng et al.], [Kapoor et al. 2022], [Vandewiele et al. 2022].

6.4.2.5.2 Lack of normalization

Optimization methods used in machine learning often assume normalized inputs and often do not perform well on un-normalized data. Another common pitfall is the use of different normalization methods or normalization parameters between training and serving. For instance, with the Min-Max normalization, the bounds computed with the training set are used identically for the test or serving data, and not recomputed on the test set.

6.4.2.5.3 Data leakage

In supervised learning, contamination of a model with information that is not available in production can seriously impact the model's ability to generalize and gravely distort reported results. Leakage can occur column- or row-wise. While splitting of data into multiple subsets is common practice, data leakage is still a frequent issue. Complexity in the dataset (such as hierarchical or time-series structure of training data) can be considered when testing for leakage as well as the specific order of steps in the machine learning pipeline (e.g. leakage due to

over-sampling prior to splitting). Data leakage can occur in non-obvious scenarios where test-set information is used implicitly during training and is thus a wide-spread issue. Common symptoms include suspiciously good performance metrics combined with low performance in productive systems (note that similar symptoms can be caused by other issues such as not compensating for missing input data).

6.4.2.5.4 Lack of familiarity with data

The ease with which models can be trained could tempt users to neglect exploratory analysis of the dataset. This can lead to issues such as failure to detect important edge cases, failure to adequately consider imbalanced datasets, failure to detect incorrectly labelled data or incomplete coverage of input domain with respect to space application and/or environment.

6.4.2.5.5 Data mismatch

When training and validation (or test) data come from different distributions this can lead to varying performance per dataset which could be incorrectly attributed to a high variance issue. Different data distributions can arise in various circumstances such as training data being simulated or sensors used for model training and validation differ from sensors used in production.

6.4.2.5.6 Data drift

This issue is very similar to the previous point but often encountered in online systems where shifts in data generating distribution are frequent. Symptoms include degrading performance of the live system over time.

6.4.2.5.7 Incorrect application of model metrics

A wide range of metrics exist, and their differences may not always be clear to practitioners. Care should be taken to select a suitable metric, one that is unbiased and appropriate for the relevant dataset, model, and application.

It is recommended to conduct exploratory data analysis to identify the most suitable metric. For example, accuracy might not be appropriate for a classification problem if the dataset is imbalanced. It is also recommended to consider the model and end application to find a metric that meets the specific requirements. For instance, if the goal is to minimize false positives in a classification problem, focus on precision. If the aim is to minimize false negatives, focus on recall. These examples illustrate how the choice of metric can impact model performance evaluation, but they are not intended to be an exhaustive analysis. For more information, you can refer to specific documentation on this topic.

6.4.2.5.8 Time traveling data

A special case which is recommended to avoid when dealing with time series data, is to mix-up the order which the data is represented when training a model. In machine learning applications, the use of "time-traveling" data, or data that is out of sequence or does not respect the order in which it was collected, is generally not allowed. This is because the order in which data is collected often corresponds to the order in which events occurred, and ignoring this order can result in inaccurate or unreliable models.

To ensure that the order of the data is respected, time stamps on the data can be taken into account when training machine learning models. This includes information about the past that might have been received later.

The time stamps allow the data to be properly sequenced and can be used to ensure that the models are trained on the correct data. For example, if a model is being trained to predict future events based on past data, it is important that the past data are properly ordered based on the time stamps to ensure accurate predictions.

It is important to note that violating the time ordering of data can also result in "data leakage," where information from the future is inadvertently used to make predictions about the past. This can lead to overly optimistic performance estimates and inaccurate models.

6.4.2.5.9 Data bias

Data bias refers to the presence of errors in the dataset, such as underrepresented data, overrepresented data, or inaccurate data. These errors can lead to incorrect, unfair, or inaccurate results. There are different types of data bias, including confirmation bias, sampling bias, selection bias, or cultural bias, to name a few.

For instance, when collecting data, it may not always be possible to obtain an equal number of training examples for each class of the input domain. Some classes may be underrepresented or missing entirely, leading to selection bias. Training data that does not sufficiently cover the input domain can result in out-of-distribution cases in production. Therefore, the ML model learns to interpolate the area of the provided input domain but may fail to cover the complete input distribution. This type of bias can be addressed by means of data augmentation or by gathering new data to increase the representation of the underrepresented class.

Performing an exploratory data analysis, as mentioned in section 6.4.2.5.4 can help to identify the presence of bias in the dataset.

6.4.3 Machine Learning Model Testing

6.4.3.1 Model Testing

6.4.3.1.1 Overview

Performance metrics can be used as the main performance indicators during the model development phase. Performance metrics against the training and validation datasets help us to identify the best candidate model. The same performance metrics against the test dataset allow us to evaluate the behaviour of the model under unseen data.

Model testing aims at improving the model trustworthiness, by applying different methods, to complement the model evaluation based on performance metrics. Most of these testing methods can be applied to an already trained model while others can be included in the training phase already.

The structure of the chapter is the following. First the concepts of operational scenario and operational design domain (ODD) are introduced in the context of model testing. Then different methods to test the model are presented, with respect to the ODD and the model quality characteristics introduced before in section 6.4.2.3 “Model quality characteristics”. Lastly, Explainability is discussed in one dedicated section.

6.4.3.1.2 Testing context

The concept of operational scenario and operational design domain, which were already introduced in the previous section, setting the context of data representativeness, can also help to set the context for the model testing process. Requirements allocated to the AI/ML model can be derived from the operational scenarios and operational design domain definition. The following are some of these requirements relevant to the model testing process that are taken from the [EASA paper].

- functional requirements allocated to the AI/ML model
- safety requirement allocated to the AI/ML model
- operational requirements allocated to the AI/ML model, including ODD.
- non-functional requirements allocated to AI/ML model (e.g. performance, scalability, reliability, resilience, etc.)
- interface requirements

These requirements can be used to define the context and the scope of the model testing.

6.4.3.1.3 Testing methods

6.4.3.1.3.1. Specific examples testing

Requirements allocated to the AI/ML model can define specific outcomes and performance thresholds on a subset of specific examples under specific operational scenarios. The model can therefore be tested on a test set of these specific examples or corner cases to ensure compliance.

This testing approach follows the concept of slicing testing. In slicing testing a subpopulation of interest it is identified and a subset from the test dataset is created. If the test dataset was not representative enough for this subpopulation new data can be obtained to improve the representativeness of the test subset. Then, the model is tested on this new subset and performance metrics are obtained and evaluated.

Different criteria, whether technical or business-related, can be used to define the subpopulation of interest, and eventually it depends on the application at hand. Below two main criteria are initially identified:

- Input's importance: Underperforming conditions associated with specific inputs can have a bigger impact on the overall performance of the AI/ML model. A subset can be created, containing only these specific inputs, and performance metrics obtained and evaluated. For instance, a multi-image classifier model is trained to detect different types of hazards for a collision avoidance system. Different risk levels, depending on the associated risk

to the identified hazard, are defined. A subset containing only inputs identified as high-risk can be created to evaluate the performance of the model on those specifics' critical inputs.

- Specific ranges of the ODD: A subpopulation, from the original operating parameter's range defined in the ODD, can be identified as more critical or problematic. A subset can be created, containing only input that meet these specific operating parameter' range, and performance metrics obtained and evaluated. For instance, a model is trained to perform image classification under a certain range of luminosity conditions. In order to better evaluate the performance of the model under low range of luminosity conditions a subset can be created to only contain images from the lower luminosity spectrum of the operational range.

As a result of this testing approach, if underperforming situations are identified, actions can be taken to improve the model performance. For instance, when possible, new training data can be obtained, following the distribution of the identified underperforming examples to retrain the model.

6.4.3.1.3.2. Neural Network coverage testing

In a similar vein to branch and decision coverage in traditional software, neural networks can be subjected to coverage metrics, offering a proxy for the effectiveness of the test dataset. This can lead to the automated generation of test cases: new inputs transformed from the original data (in the case of image data, either through pixel-value transformations or affine transformations) with the aim to optimize some test adequacy criteria. The metrics used to identify neuronal coverage is based on the idea of measuring the proportion of neurons activated in a neural network given a test set as input. Below is a summary of tools and techniques that provide ways to perform neuronal coverage:

- DeepXplore offers an automated white-box testing approach for deep learning systems that provides neuronal coverage metric for a neural network that uses the ReLU as its activation functions. In this approach, a test suite is said to achieve full coverage if for every hidden unit in the neural network, there is some input for which that hidden unit has a positive value (the neuron has been "activated") (Pei, 2017).
- Tensorfuzz uses coverage-guided fuzzing, an existing technique from traditional software engineering, and adapts it to be applicable to testing neural networks. (Odena, 2018). TensorFuzz is particularly useful in its ability to surface disagreements between models and their quantized versions.
- DeepEvolution builds upon neuronal coverage and offers a search-based testing approach with local neurons coverage (new neurons covered by the mutated test input that have not been covered by its corresponding original test input) and global neurons coverage (new neurons covered by the mutated test input that have not been covered by all the previous test inputs, including both genuine and synthetic test inputs) as metrics (Networks, 2019). Deep-Evolution proved better suited at detecting disagreements between models and their quantized versions than TensorFuzz.

6.4.3.1.3.3. Out of distribution testing

The model can be tested on a test set that contains Out-of-Distribution (OOD) examples to see how the model behaves on unpredictable data. OOD examples are those that are drawn from a different distribution than the training dataset. This testing approach is also relevant at system level. When done at system level this approach allows to see if any back-up systems implemented (non-ML safety nets, rule-based encapsulations of the ML systems) correctly deal with known-unknowns (see 5.1.4.2.2 for more details). This level of testing can be automated through scripts; the scripts will have to be validated and verified.

6.4.3.1.3.4. Augmented data set - Noise

Noise can be added to the training data set to improve generalization of the model under its presence. The type of noise to be added depends on the specific application. If the expected noise during operations can be modelled, then the data can be augmented following that model. This is generally the case for payload data, for which several noise models are available throughout the development process (committed, expected, worst-case). Considering ageing effects on the expected noise (model) can also be considered in the testing strategy.

In the case where the operating noise is unknown other types of noise can be used, depending on the type of inputs, such as: Gaussian noise, salt and pepper noise or speckle noise.

6.4.3.1.3.5. Adversarial testing

Different taxonomies on adversarial attacks exits, at higher level they can be classified as: white-box attacks, grey-box attacks, or black-box attacks, depending on the degree of knowledge the attacker has on the system. At lower level they can be classified in poisoning attacks, evasion attacks or model extraction attacks. One common characteristic of these attacks is that they try to force mistakes or misbehaviour on the model's prediction by using altered inputs called adversarial examples. Adversarial testing consists of evaluating the vulnerability of the model, by feeding it adversarial examples, and evaluating the prediction. If the model is found vulnerable to adversarial attack different mitigation methods can be applied, such as: adversarial training, input regularization, gradient masking, defensive distillation, defensive dropout, or by providing a mechanism that can detect and clean deceptive inputs. It is out of the scope of this document to provide a detail description on the different type of attacks, adversarial testing methods and mitigation techniques. However, it is important to mention that adversarial testing can be considered per application, evaluating the feasibility of each attack and taking measures to mitigate their impact when vulnerabilities are found in the system.

6.4.3.1.3.6. Formal Methods and Mathematical Verification

Formal methods refer to mathematically-based techniques that are used to analyse certain properties of a model that can be proven mathematically in a formal manner. Examples of formal methods include abstract interpretation, property checking, symbolic execution and other. Different formal methods can

be applied for ML models to analyse local robustness, robustness to adversarial attacks, model stability, model execution timing.

This approach was shortly discussed earlier in the context of *confidence*, as this method is to mathematically prove that the ML model, based on type of algorithm and/or the data solution space covered by the model, is enough to exclude certain fault modes from happening. When achievable, this type of formal mathematical verification avoids the architectural changes of additional software/hardware for monitoring and limiting the functionalities. Formal verification requires a strict and extensive method:

- To formalize the properties to secure,
- To identify the most suitable technique with regards to the complexity of the problem,
- To use tools with good maturity
- To properly handle scalability of complex systems, despite the state space explosion problem.

A trade-off between formal methods and safety cages options is recommended to weight the pros and cons of both paths, as neither of them is simpler by default.

This topic is considered ongoing research within the field of machine learning; not only does it requires strong algebra background to apply, but verification currently becomes difficult as soon as the model is non-trivial / academic, e.g. even for a shallow neural network type models with elementary layers. Besides, mathematical doesn't mean analytical: some mathematical verification may require much computational power, even higher than training the model. It equally benefits from the efficiency of GPUs for this task. Therefore, it is generally recommended to use these methods with care, until they are more widespread, and complement them with sampling techniques to confirm the conclusion.

Formal methods in machine learning is an active research area and can be evaluated and applied on cases-by-case basis depending on relevance for a specific project. For example, usually formal methods may require a lot of computational resources and may not be practical to apply for large machine learning models. Examples of research studies on ML formal methods include [Gehr et al. 2018] [Singh et al. 2019] [Cheng et al. 2017].

To keep-up with on-going practical advances in this field, the reader will find interesting information in the proceedings of the Workshops on Formal Verification of Machine Learning (WFVML). Research has been especially active to cover 3 types of verifications:

a. Bounding of the output space: given an allowed hyperspace χ included into the possible input space of the model f, compute a minimized output hyperspace encompassing all the possible values of $f(\chi)$; the verification is achieved if this bounding area remains inside the safe zone of the system. See also illustration in Figure 6-9.

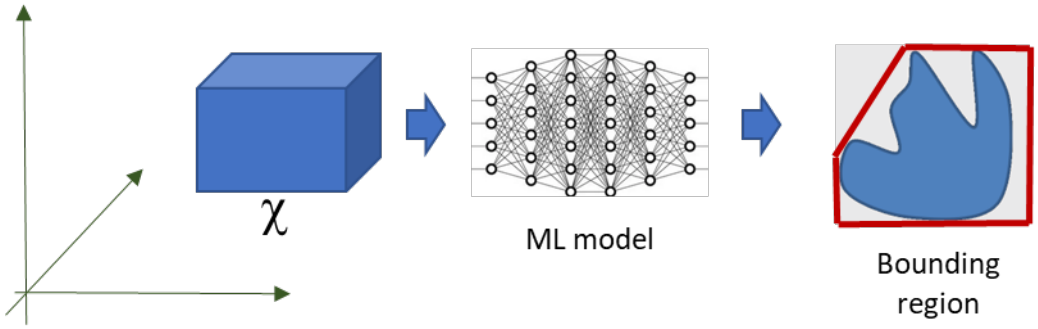

Figure 6-9: ML Bounding Region

b. Robustness to input variation: by extension of the previous case, for a ML model driving the system as a validated classifier, the output classification is not degraded if any of the inputs of the allowed hyperspace χ is affected by a perturbation δ. In Figure 6-10 below, despite the uncertainty δ in the input parameters, the resulting uncertainty on the output space does not cross the partitioning rule on which the correct behaviour of the algorithm has been validated.

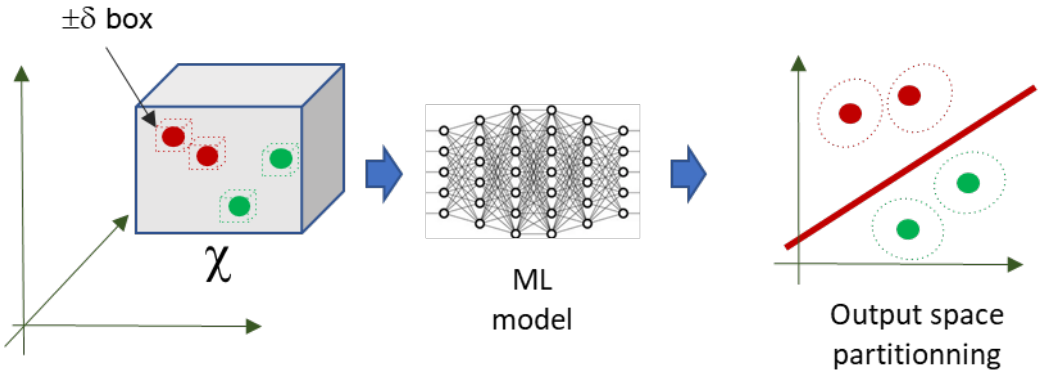

Figure 6-10: Robustness to input variation

Both problems can be seen:

- Either as constrained non-convex optimization tasks, that can be solved thanks to several convex relaxation options (altering the activation function), so that the problem can be addressed by semidefinite relaxation (see [MATHVERIF1]), or linear programming (see [MATHVERIF2])
- Or as the resolution of a dynamic system where each time step would be a progression through a layer of the ML model. Hence it can be interpreted as a linear system with non-linear feedback (i), or as a linear system quadratic wise constrained (ii):

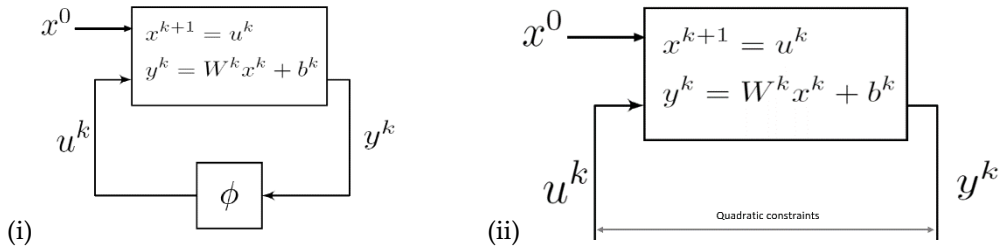

Figure 6-11: Dynamic system interpreted as non-linear feedback (i) or as a linear system quadratic wise constrained

Once expressed as quadratic constraints:

$$\begin{bmatrix} x^k \\ x^{k+1} \\ 1 \end{bmatrix}^T \cdot M^k \cdot \begin{bmatrix} x^k \\ x^{k+1} \\ 1 \end{bmatrix} \geq 0$$

the problem can be resolved using the techniques of semidefinite programming (see [MATHVERIF3]).

c. Robustness of neural network driven closed-loop system, see Figure 6-12, when subject to disturbances and uncertainties:

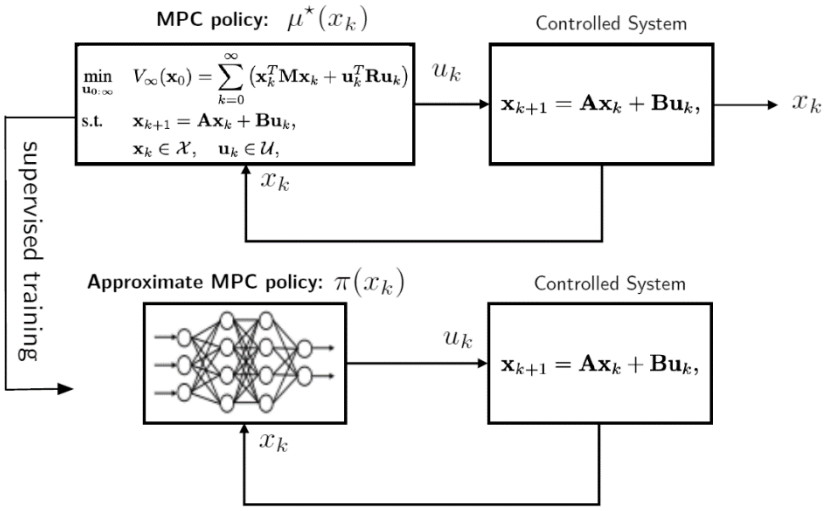

Figure 6-12: Robustness of neural network driven closed-loop system

For this kind of ML algorithms as well, both semidefinite relaxations (see [MATHVERIF1]) and semidefinite programming (see [MATHVERIF3]) as mentioned in the previous cases are relevant to verify the behaviour of the ML algorithm.

6.4.3.1.3.7. Statistical Testing

Statistical testing relies on the generation of test data according to a probabilistic model, in order to assess the likelihood that the observed behaviour of the system matches the specification with a certain confidence level. In general ML models

and their input domain are highly complex so that it is not practically possible to apply equivalence classes testing approach. Statistical testing methods include for example traditional Monte Carlo sampling approaches and more advanced Markov Chain Monte Carlo with subset simulation that aims to generate samples to get closer to a specific area of interest such as a failure domain for machine learning components. Current research aims at utilizing this concept to support the identification of suitable and sufficient number of test cases for machine learning components [Au et al. 2014] [Schwaiger et al. 2022].

6.4.3.1.3.8. Detection of unintended behaviour

ML models complexity and obscurity can result in model unintended behaviour that might not be revealed by requirements-based testing. For example, transfer learning techniques can cause residual functionality of ML models irrelevant to the intended operational domain. Mitigation measure that can address the unintended behaviour include:

- Gradient-weighted class activation mapping or saliency analysis to identify inputs causing unexpected response
- Out of distribution detection to ensure operation in the intended domain
- Architectural mitigation (safety cage)
- Transfer learning avoidance

6.4.3.1.3.9. SEU testing

Bit-Flip Attacks (BFA) on model parameters can be used to simulate the consequences of SEUs on the model's performance. Mitigation techniques can include techniques that are already used in space systems such as: voting systems or ensemble methods.

Figure 6-13 associates each method with the most relevant model quality characteristic.

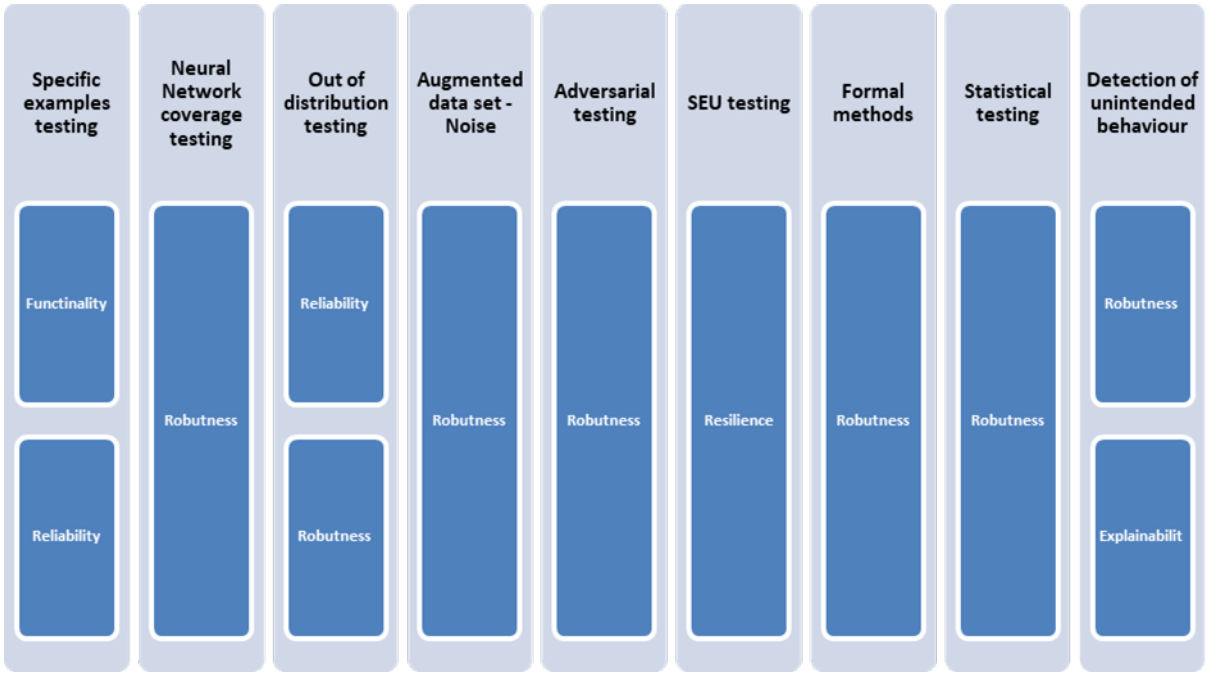

Figure 6-13: Model Quality Characteristics

6.4.3.1.4 Post-Training Optimization Testing

The goal of post-training optimization is to adapt to the final target architecture with the best compromise in terms of performance degradation; it is not to improve the raw performance that should have reached its reasonable asymptote in the steps before.

Trained models can be subject to post-training optimization methods. There are different optimization methods including, but not limited to, pruning, quantization, model knowledge distillation, half-size floating point operators, simplified activation functions, operator/layer fusion or deep learning compilers. These methods are especially relevant when on-board execution of AI is identified as the operational environment and therefore the resources associated with the final embedded HW are limited compared to the available ground HW. Different execution performance aspects are affected by the optimization, being the main ones: inference/execution time, throughput, power consumption and memory footprint.

Note that in the case of on-board execution, optimization techniques can be necessary to allow the deployment of the AI software in the target embedded HW. In the case of ground segment, optimization techniques can still be applied to optimize some of the execution performance aspects mentioned before, but are, in general, less critical. Note that there is always a trade-off to be considered between the level of optimization on the execution performance aspects and the associated drop in the AI performance metrics.

For the above reason when an optimization method is applied to a trained model the resulting optimized model can go again through all the evaluation and test steps defined before:

- The optimized model can be evaluated against the test dataset and AI performance metrics obtained. The objective is to ensure that the drop in performance associated with the optimization, if any, it is acceptable.
- The optimized model can be tested against the different existing test sets as defined in section 6.4.3.1.3 "Testing methods".

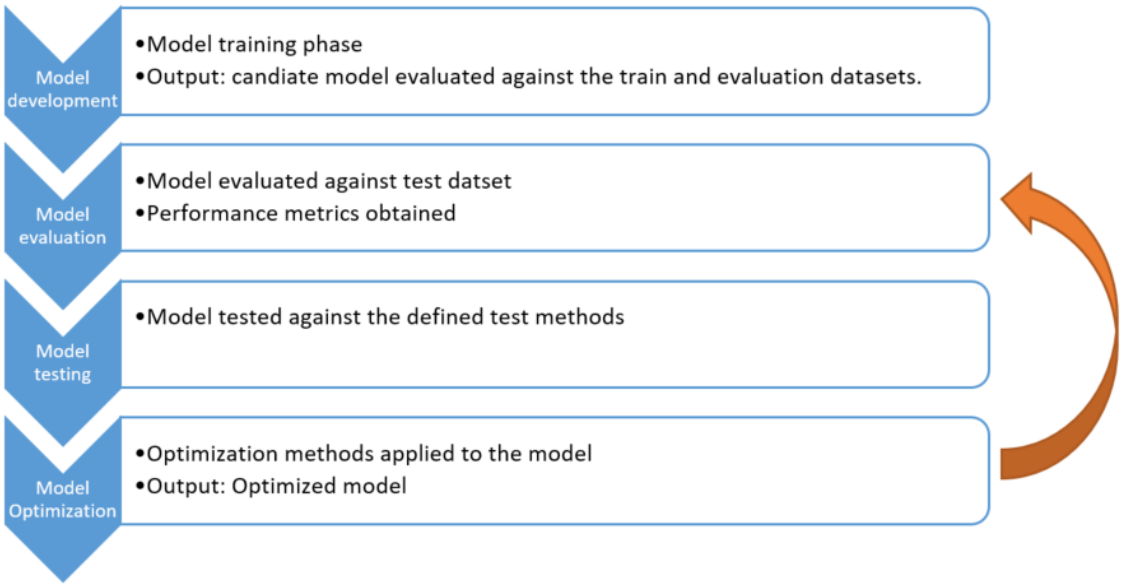

Figure 6-14: Post-Training Optimization Testing

If underperforming or misbehaviour of the optimized model is detected during the evaluation or testing phase it is recommended to take different actions to correct the situation:

- Trying different optimization techniques: Depending on the specific constrains and optimization objectives, different optimization techniques can be applied. For instance, if the objective of the optimization is to reduce the size of the model, but no constrains are identified regarding the use of float-point data types, pruning can be evaluated as post-training optimization method over quantization. Alternatively, some machine learning frameworks allow aware training optimization methods, such as quantization-aware training. This can be an option, especially when post-training optimization techniques have an unacceptable impact in the model performance. Quantization-aware training can lead to optimized models with lower performance impact, at the cost of a more intensive and demanding training process.
- Going back to the model development process: If no optimization method is found that yields an acceptable performance one might need to go back to the model development process and try more drastic approaches:
  - Try different architectures: Try smaller architectures or architectures specifically design for embedded use. While these architectures might yield worse performances before quantization, compared with more complex one, they might be less impacted by

optimization methods or might not require an optimization step prior deployment in the embedded target.

– Try retraining the model with new data: If underperforming of the optimized model can be identified to specific data input distributions, it is recommended to evaluate the feasibility of improving the training dataset targeting that input distribution. This alternative will depend on the specific data availability as it might not be possible to get new data to improve the training dataset.

Finally, optimization methods can be applied on the development machine or in the final target HW, this depends on the framework used to optimize the model and on the final target HW type. If the optimization is applied in the development machine, the optimized model can be initially evaluated and tested in this same machine, but it can be evaluated and tested again when integrated in the final target HW. The reason is that some metrics are HW dependant and cannot be evaluated in the development machine, such as: inference time or power consumption. This process is called target specific testing and it allows to:

- Check the compatibility of the model with the target HW and execution platform, for instance quantization, network support.
- Check the model performance in the target HW: inference time, throughput, memory, power consumption.
- Reconfigure inferencing platform based on performance/implementation needs:
  - Increase/decrease parallel processing of the model (configure parallel threads).
  - Optimize the model architecture.
- Apply other optimizations such as resource sharing or target specific resource utilization

6.4.4 System Testing

6.4.4.1 Overview

In this section, the topic of testing, verifying and validating systems that use machine learning as a component is examined, and guidance on best practices for ensuring their reliability and safety is provided.

As machine learning is increasingly being integrated into various systems and applications, it is essential to ensure that these systems are properly tested, verified and validated. A failure in the machine learning component of a system can have severe consequences, including safety hazards, financial losses, or damage to the reputation of the system and its developers. Therefore, the reliability and safety of ML-based systems must be thoroughly evaluated before deployment.

The structure of the chapter is the following. First a definition of AI-based systems are provided together with classification of AI applications. Then follows the presentation of a proposed verification and validation process for AI-

based systems which includes ML models. Lastly step by step methods and tools are presented both for performing the analysis of failures for said system (FMEA/FMECA), but also how to mitigate the failures using safety cage architectures.

6.4.4.2 Overview of ML on system level

6.4.4.2.1 Overview

When the application of Machine Learning is discussed, an emphasis is often placed on the underlying algorithm, or on the data used for training and validation of the ML models. However, if ML is to ultimately be implemented as part of any space system, in similar fashion to good systems engineering practice, the interactions between the ML software solution and the surrounding systems have to be analysed, in an attempt to uncover failure modes which could represent an unacceptable risk to the system. Hence, such analysis is recommended to be one of the driving efforts in verification and validation of ML for space applications, to find potential mitigative actions which can bring the risk down to an acceptable level.

When taking the systems engineering approach to verification and validation, performing such activity for ML -based solutions, allows us to follow the already existing general approaches for building and qualifying space systems, including software, where micro-environments (individual sub-parts and components) are analysed with respect to interactions with macro-environments (overall subsystems and systems).

The verification and validation need of a space system is normally already identified during the solution design phase where mission requirements are analysed with respect to planned system performance, which then can be used to define a system architecture for the mission, for which suitable sub-systems and sub-components are chosen, including software such as ML. It is therefore recommended only once the mission requirements are available, to define what is acceptable (or undesired) performance (for example limiting false positives), and not isolated focus purely on the models performance on the data, when evaluating the performance of Machine Learning applications for space systems and missions. As these topics tend to be more related to model evaluation, verification and validation, they will not be discussed further here, but it should be understood that if such mission requirements cannot be satisfied on data and model level, then it might be possible, or needed in case of having a connected risk, to be handled on solution and system level. A specific example of such is Autonomy, as discussed in section 6.4.4.2.3.

In the following section, the acceptance of a space system using ML is discussed, and what is proposed is to look at ML part as a black-box component, which then allows the verification and validation of a proposed solution much similar to normal systems engineering verification and validation of software. 6.4.4.3.3However, as it will also be elaborated, certain additional aspects have to be considered when dealing with the stochastic nature of the ML models.

6.4.4.2.2 AI-based System definition

To allow the further discussion of Machine Learning on system level, the definition of a ML- and likewise AI-based systems, as per the [EASA paper] and can be described as following:

- An AI-based system (a.k.a. ML-based system) is composed of several traditional subsystems, and at least one of them is an AI-based subsystem. This system can include hardware and software.
- An AI-based subsystem (ML-based subsystem) embeds at least one AI/ML constituent.
- An AI/ML constituent is a collection of hardware and/or software items (including the necessary pre- and post-processing elements), and at least one specialised hardware or software item containing one (or several) ML model(s).
- Within the ML constituent, some form of ML model is usually found. It can refer to directly, or indirectly as a part of the software connected to the ML model/models (for example in case of an application), which can then be referred to as ML-based software component, ML-software or ML application. Additionally, an AI-based system can of course also include traditional hardware and software items that do not include an ML inference model and can generally be qualified independently, as long as they are not within the AI/ML constituent which is indicated as the area which might be affect by the output of the ML model(s). How to find the line between what is included in the AI/ML constituent, and what is not is discussed further is in section 6.4.4.3.3 on FMEA/FMECA.

It is here assumed that the AI-based systems are containing some form of machine learning, which introduces the stochastic nature into the design of the system, and in some cases into the system itself. Although this does not imply that it is solely ML. Moreover, as discussed in the upcoming subsection 6.4.4.3.4.2 on Safety cage architecture, AI hybrid solutions are often created to allow for a more dependable system, e.g. as part of a safety cage.

However, as it will be discussed in section 6.4.4.3.3, based on the interfaces and types of interactions between the parts within a given AI-based system that contain ML, the FMEA/FMECA and potential HSIA are derived, which will provide the overview of needs that have to be meet for a solution acceptance.

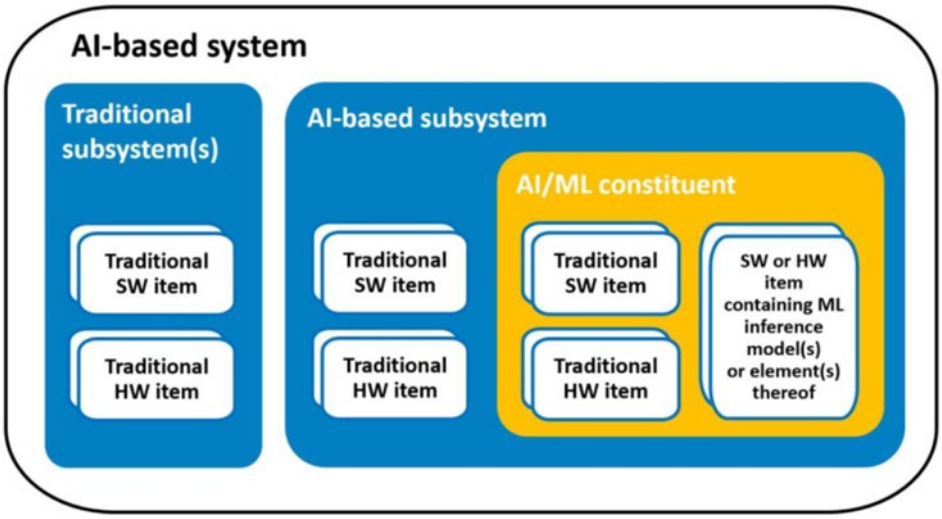

Figure 6-15: [EASA paper] Decomposition of the AI-based system

6.4.4.2.3 Classification of autonomy for AI applications

In the previous section 6.4.4.2.2 a definition of an AI-based system was provided, but another topic that is worth considering on system level is the concept of operations (ConOps) with respect to the unique characteristic often associated within the usage of artificial intelligence, autonomy.

ConOps should here be understood as the high-level statements which explains the characteristics of a system from the user's point of view. This would ultimately be the statements which describes exactly the "what, how and why", for the usage of an AI-based system, which can then be further broken down into mission and system requirements. One of the unique features of the "how", which AI especially makes possible, is the opportunity to create autonomous decision making.

One of the areas which particularly comes to mind when discussing the need for automatization and autonomy to create a higher efficiency is space operations. All of the operations during a space mission rely on planning and monitoring, performed significantly in advance of actual operations, and is most often relying on the work and actions of humans. Here, only some action might be automated, such as calculations of certain parameters, but most things need to be initiated by a human, and ultimately most, if not all, decision making is done by a human. With the developments within Artificial Intelligence, a re-distribute of responsibilities between the ground segments and the spacecraft, introducing more automated operations, can be introduced, which can reduce operation costs, increase scientific return and support future missions beyond LEO where significant time delay is present. Likewise, AI can also support operations of ground station allocation and management with advanced scheduling tools providing flexibility in allocation, mixed-initiative approaches, what-if analysis, multi-objective optimization and automation. Examples of such ConOps where autonomy would be relevant can be mentioned the operation of large constellations or for vehicles such as planetary rovers and space probes, where there might be a signification delay in communication to the ground stations.

Of course, there are also areas where the autonomy is not as important or simply taken for granted, in the sense that we automatically assume that the autonomy

is there, without a need for further evaluation, as no real decision was taken in the frame of an operation-like scenario. Of such areas can be mentioned using AI for Data transformation or compression, where the algorithm "intelligently" performs the trained function independent of a human (autonomously), but the function is seen as independent from any application where the consequences of wrong doing is high enough that it can be considered whether a person should be in the loop, for final evaluation or decision making.

In summary, what could ultimately be considered is that the level of autonomy is a measure of how independent the system is of a human for decision-making, and based on the level needed/chosen, the risk and complexity of said system might in turn heighten. This could be linked to the consequence of the AI-based system making a wrong decision, and whether a man-in-the-loop can mitigate the risk, by e.g. evaluating the suggestion of the system before it is implemented. Section 6.4.4.3.3 takes the reader through the suggested FMEA/FMECA analysis, which is here recommended as the analysis for evaluating the AI-based system compared to internal and external interfaces and potential failure modes. However, it is additionally recommended that the designers of an AI application, could consider the possible consequence of the application being wrong. And, as discussed in the subsection on Safety cage building blocks and tools, the man-in-the-loop is only one of the tools which can then be used to mitigate the risk and consequences if found too severe.

Additional considerations here include the general assumption that, unless an AI-based solution is specifically designed to be part of a man-in-the-loop system, e.g. in the case of co-bots or cognitive assistants for decision support, the aim is generally to have as high a level of autonomy possible within acceptable risk, in order to create solutions that are as independent and effective as possible.

Based on the definition in the [EASA paper], three general levels of AI autonomy have been identified, as a means of classifying the AI-based systems and applications. This scheme has been proposed based on prognostics from the Aeronautics industry regarding the types of use cases foreseen by AI-based systems, which is similar to what is found in the Aerospace industry. Figure 6-16 shows the three scenarios of staged approaches for the deployment of an AI applications with different levels of conjunctions with a human, starting with assisting functions (Level 1 AI), human-machine collaboration (Level 2 AI) and at lastly higher autonomy of the machine (Level 3 AI).

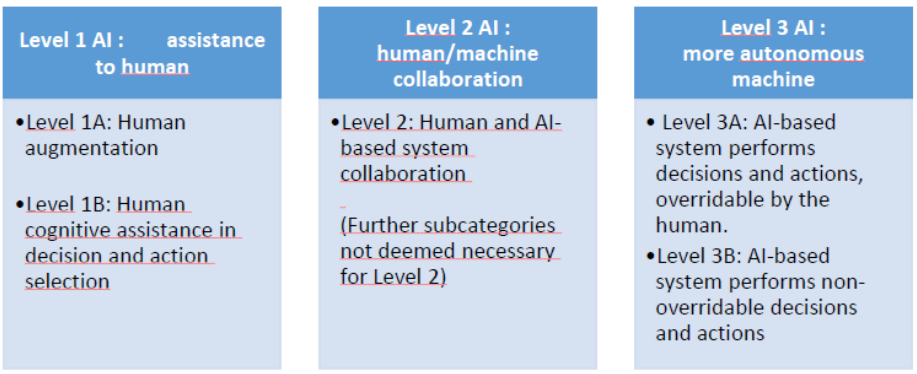

Figure 6-16: [EASA paper] Classification of AI applications autonomy

Note: The topic of autonomy will not be handled more in depth within this handbook as it is an ongoing research topic, hence the scope presented here does not do the complexity justice. For example, the level of autonomy can be further split into sub-categories based on type of function and how it is built. However, the approach presented for verification and validation of AI-based systems in the following sub-sections can still be applied.

6.4.4.3 Verification and validation process of AI-based systems

6.4.4.3.1 Overview

An AI-based system can contain both hardware and software which is either directly or indirectly coupled to an AI software component(s), see also 6.4.4.2.2. The main point which can be considered for such systems is whether the mechanisms for the generation of the output from the AI parts of the system are stochastic. The data-driven part of AI, which is the predominantly machine learning, represents stochastic models, which is the main focus of this handbook. However, one should be aware that very large rule-based systems can artificially have similar stochastic nature, if the complexity of interactions between their rules and knowledge is high enough. Therefore, the suggestion from this handbook goes as following:

If the system contains data-driven AI, such as machine learning, or any user input-driven AI which decision logic cannot be described easily by a decision tree, AI-specific processes of verification and validation have to be considered.

In the following sub-sections, the process of verification and validation of stochastic-natured AI is described.

6.4.4.3.2 Safety criticality assessment

The key driving factors on system level for the usage of ML-based systems is safety criticality. From the point of view of this handbook, AI/ML constituent can be categorized like "traditional" software in line with ECSS-Q-ST-80 as show on the Table 6-3.

Table 6-3: Software Criticality

Software criticality category	Definition
A	Software involved in category I functions AND: no compensating provisions exist
	Software included in compensating provisions for category I functions
B	Software involved in category I functions AND: at least one of the following compensating provisions is available, meeting the requirements defined in ECSS-Q-ST-30 clause 5.4 and ECSSQ-ST-40 clause 6.5.6.3: • A hardware implementation • A software implementation; this software implementation shall be classified as criticality A • An operational procedure
	Software involved in category II functions AND: no compensating provisions exist
	Software included in compensating provisions for category II functions

Software criticality category	Definition
C	Software involved in category II functions AND: at least one of the following compensating provisions is available, meeting the requirements defined in ECSS-Q-ST-30 clause 5.4 and ECSSQ-ST-40 clause 6.5.6.3: • A hardware implementation • A software implementation; this software implementation shall be classified as criticality B • An operational procedure
	Software involved in category III functions AND: no compensating provisions exist
	Software included in compensating provisions for category III functions
D	Software involved in category III functions AND: at least one of the following compensating provisions is available, meeting the requirements defined in ECSS-Q-ST-30 clause 5.4 and ECSSQ-ST-40 clause 6.5.6.3: • A hardware implementation • A software implementation; this software implementation shall be classified as criticality C • An operational procedure
	Software involved in category IV functions AND: no compensating provisions exist

Criticality categories are assigned to software products as specified in ECSS-Q-ST-30 clause 5.4, and ECSS-Q-ST-40 clause 6.5.6.3, see Table 6-4.

Table 6-4: Comparing Dependability and Safety

SEVERITY	SAFETY	DEPENDABILITY (ECSS-Q-ST-30)	SAFETY (ECSS-Q-ST-40) Extract from ECSS-Q-ST-40
Catastrophic	1	Failure propagation (requirement 4.2c.)	• Loss of life, life-threatening or permanently disabling injury or occupational illness. • Loss of an interfacing manned flight system • Severe detrimental environmental effects • Loss of launch site facilities • Loss of system
Critical	2	Loss of mission	• Temporarily disabling but not life-threatening injury, or temporary occupational illness • Major detrimental environmental effects • Major damage to public or private properties • Major damage to interfacing flight systems • Major damage to ground facilities

SEVERITY	SAFETY	DEPENDABILITY (ECSS-Q-ST-30)	SAFETY (ECSS-Q-ST-40) Extract from ECSS-Q-ST-40
Major	3	Major mission degradation	
Minor or Negligible	4	Minor mission degradation or any other effect	

As indicated from the two tables above, even software of criticality B could lead to safety critical effects. Therefore, the recommendation is to at least establish a FMEA to identify critical functions and allow for proper risk mitigations, as it will be discussed in the following sections.

6.4.4.3.3 FMEA/FMECA

6.4.4.3.3.1. Overview

Generally when dealing with space assets, due to the complexity of the several interacting systems and subsystems, as an integral part of the design process and as tools to drive the design along the project life cycle, the following two methods are proposed “failure modes and effects analysis” (FMEA) and “failure modes, effects and criticality analysis” (FMECA).

The primary purpose of the FMEA is to identify possible failure modes of the system components and evaluate their impact on safety and system performance. FMECA is the extended version of FMEA that classify potential failure modes according to their criticality, the combined measure of the severity of failure modes together with the probability of occurrence. Technically, both FMEA and FMECA can be created without knowledge of how the machine learning models functionality will be implemented, as it is more concerning interfaces and failure mode propagation.

Although the FMEA/FMECA is primarily a reliability analysis, it provides information and support to safety, maintainability, logistics, test and maintenance planning, and failure detection, isolation and recovery (FDIR) design (ECSS-Q-ST-30-02, Introduction).

A more software related version/approach is described in ECSS-Q-HB-80-03A, chapter 6.2 as SFMEA (Software Failure Modes and Effects Analysis).

This philosophy and the specific approach is also valid for Machine Learning models.

It is assumed that during the training phase of ML no safety relevant aspects must be considered especially not any which are part of the resulting system and the ML models is treated as the lowest unit of software.

6.4.4.3.3.2. Failure Mode Taxonomy

To support the identification of modes for Machine Learning, Figure 6-17 presents a failure mode taxonomy to support the creation of the FMEA.

In general, it is recommended to focus on reasonable failure modes based on the functional view (requirements), high level architecture and input/output (I/O) interfaces. Any other aspects such as environmental impacts are not addressed in this handbook because they are part of budget considerations similar to "traditional" software development.

A failure mode can be created either because of an incorrect implementation of the function or Input/Output (I/O) errors. I/O errors refer to the inputs or outputs of the unit. The most obvious I/O errors can be

- a wrong amount of input data provided to the module, e.g. not enough or too much input for model evaluation (too much, too little),
- an incorrect value provided by the previous processing step,
- I/O values outside the specified value range,
- type error if actual and expected I/O type doesn't match, e.g.
  - signed/unsigned int
  - 32-bit number instead of a 16-bit number

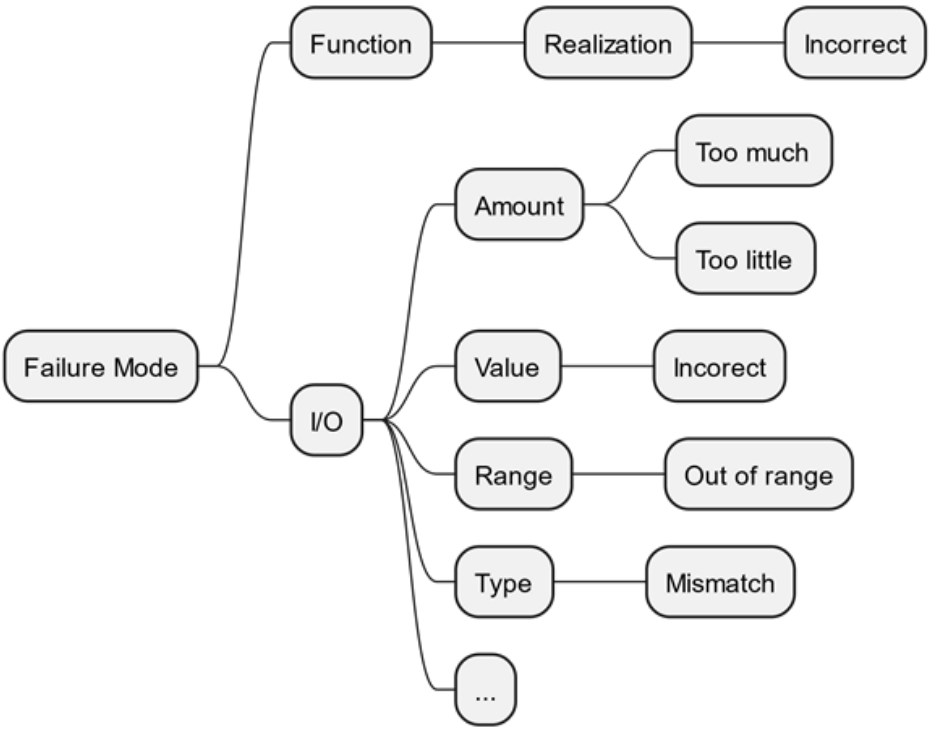

Figure 6-17: Failure mode taxonomy for Machine Learning

6.4.4.3.3.3. FMEA

As a minimum this handbook recommends to perform a **FMEA** which is intended to examine:

- each module for each **Failure Mode**,
- determine the local effects and
- the effects at the system level.

Creating a FMEA

Details and guidelines on how to conduct and create a FMEA can be found in ECSS-Q-ST-30-02C (Annex B, FMEA worksheet – DRD) and Q-HB-80-03A (6.2.2 Procedure)

It is recommended to use taxonomy path as steps to perform the FMEA in order to find all *known* possible failure modes (see Figure 6-17).

Implementing (functional) FMEA

As the name indicates in a functional FMEA the functions of the module, rather than the items used in their implementation, are analysed.

Figure 6-18 shows an example of using our defined failure mode taxonomy above.

FMEA						
Product:		**System:**		**Subsystem:**		**Equipment:**
No.	**Item**	**Function**	**Failure mode**	**Failure Cause**	**Failure effects**	**…**
		E.g. Calculate pressure	Function.Not_Performed			
			Function.Performed_Wrongly			
			Function.Performed_Wrong_Timing			
			I/O.Amount.Too_Much			
			I/O.Amount.Too_Little			
			I/O.Value.Incorrect			
			I/O.Range.Out_Of_Range			
			I/O.Type.Missmatch			
			…			

Figure 6-18: Example of a FMEA table

6.4.4.3.4 Risk mitigation

6.4.4.3.4.1. Overview

Once the FMEA has been carried out, an understanding of the failure modes should have been established. If any failures are of Severity level 2 or higher, it should be considered what mitigative measures can be done. For this please also look at Q-ST-30C Rev1, chapter *5.4.2 Assignment of software criticality category*. To

avoid common cause/common mode failures, it is recommended that one ML instance cannot be used to compensate for another ML instance.

6.4.4.3.4.2. Safety cage architecture

A more conservative method, compared to the mathematical verification, is the usage of safety cage architecture. This method allows the application designer to look past many of the ML specific challenges and focus on lowering or eliminating the potential likelihood of the identified failures from the FMEA/FMECA to occur. This is done by ensuring that there are no to limited interaction between the ML model and other components/systems which could lead to an error mode. In other words, the ML model/AI application is placed inside an architecture to ensure that we can block any non-wanted interactions.

However, the art here is to create an architecture which is neither too tight, to avoid limiting the functionality of the AI application, but also not loose, to avoid any faulty output that could lead to a failure, to exit the AI application. And there is not only one way of creating such architecture, as it will be discussed in the subsection on Safety cage principles.

The definitions and principles described in this paragraph are strongly inspired by [SMOF].

Safety cage definition

The safety cage, also known as a safety strategy, is a set of safety rules to ensure a set of safety invariants, designed to abort all paths to the catastrophic states.

As indicated from the figure, the safety invariant is a sufficient condition from which it is possible to still avoid a hazardous situation, however, whose further violation would lead the system to a catastrophic state. Example could be the temperature of a component which is heated up above recommended the range for optimal performance, but if the temperature keep rising it could lead outside of the non-critical range, and ultimately have a critical impact. To avoid this, safety rules are defined, which consists of a condition and a scripted behaviour of the system and is linked with safety interventions. A safety intervention is the ability to perform system monitoring with sufficient means and cadence to prevent it of violating a safety invariant, if a set of preconditions are fulfilled. Its intention is to cut the path from safe state to catastrophic state of the system.

These interventions can be divided into inhibition (prevention of change in system state) and action (forced change in system state). The different types of interventions are discussed further in practice in the subsection on Safety cage building blocks and tools.

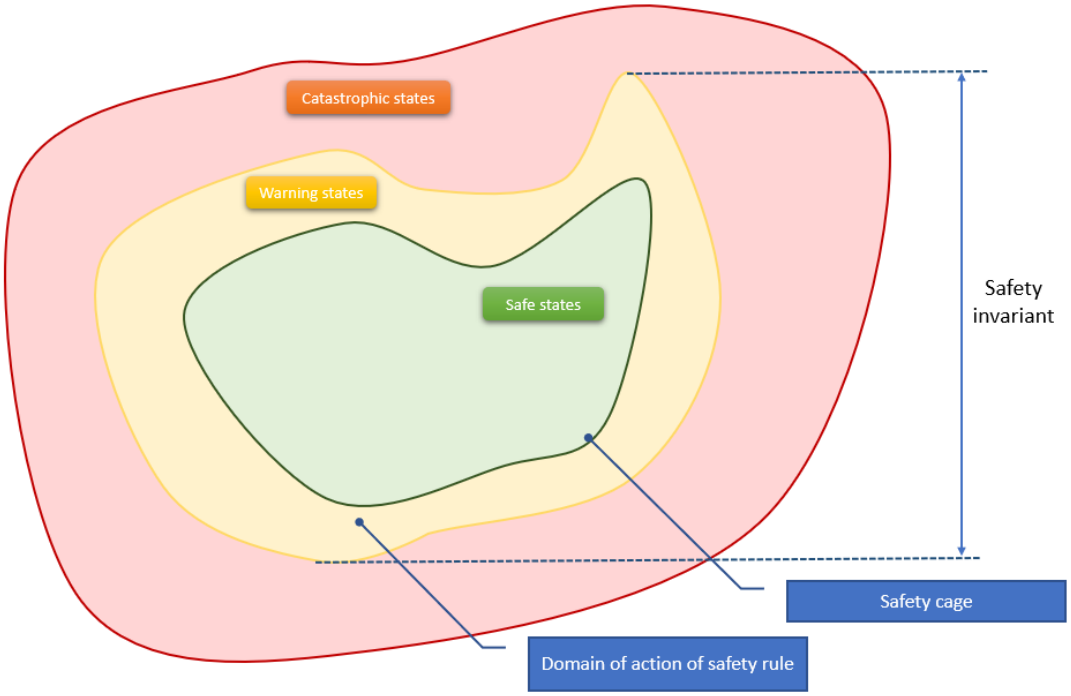

Figure 6-19: Simplified representation of the safety cage, warning states and catastrophic states within the solution space

Safety cage principles

In the previous section, the high-level definition of the safety cage was provided. In this, and the following sections, more practical advice for the implementation of a safety cage architecture is provided.

As mentioned earlier, the principle of safety cage is to intervene in the usage of the AI model output, as soon as this output exceeds the bounds of a safe domain, before reaching a catastrophic domain. This can be done either by the usage of additional software, hardware of a combination, with the choice intervention type relying on what is possible/optimal to be implemented as part of the overall AI-based system/application, and by answering the following questions for each of the different failure modes identified in the FMEA/FMECA:

a. What types of intervention can be done to avoid this type failure from propagating?

b. What rules needs to govern the interventions, including monitoring to create a functioning (verifiable) system

c. What intervention is the least limiting for the AI application, and is the easiest implemented as part of the overall application/system architecture?

By going through the above questions for each of the failure modes from the FMEA/FMECA, it should be possible to come up with individually strategies for the different failure modes.

An example for such an evaluation could be as following:

Problem description:

In the case of an AI application that estimates the distance to the surface of the Moon, to assess when the Lunar lander should fire its boosters to slow down for the landing.

Failure modes:

Here a possible failure mode could be that if it fires too late (a false negative), it will crash land, likely causing an earlier "end of mission". On the flip side, if it fires too early (false positive), it will potentially use more fuel, prolonging the descent, but if enough margin of fuel available, and time enough to correct other potential consequences of the too early burn, this should theoretically not have the same major impact on the mission execution. All these information should be clear based on the FMEA/FMECA analysis. There could of course be many reasons for the failure to take place, but likely it is due to some form of Input/Output error (for example the failure of the sensory for measuring distance) or a functional error of the AI model (e.g. it is trying to estimate the distance, however, part of the values for the input data is underrepresented in the dataset used for the training and testing of the model, i.e. outside of the known input data space, and the models resulting inference came out wrong).

Mitigation:

a. Based on the above description, it can be assumed that the main issue to solve is the case of "false negative", i.e. the crash landing, as this failure likely would be a severity category of critical. The question to first answer is what types of intervention can be used for mitigating this type of failure propagation. Of options it could be decided to use additional HW, for example having a redundant sensor instrumentation to measure the distance to the surface. If it is here assumed that the mission is using LIDAR, it could be proposed that there would be one or two redundant systems, also using LIDAR, or even that the Lunar lander would use a visual camera as well for a different source of information. Another option could be the use of additional SW, for example by having additional algorithms for estimating the distance, this could be an additional ML algorithm, but to avoid adding additional stochastic models to the system it might be preferred to be a classical algorithm for estimating the distance. However, another type of SW which could also be added would be an algorithm that compare if the output of the ML model for new incoming data is within the expected value distribution for similar data which the AI models have been trained and tested, to understand the reliability compared to the instantaneous ingoing data. Such a method would likely have to also consider the time aspect of data points perceived timewise earlier that the given "false observation".

b. With the above potential tools suggested, it is clear that we could make the AI application more robust, by placing both action and inhibition-based interventions, where the case of the HW update, we would minimise the chance of failures due to issues equipment, but this would also imply that software have to be created which monitor the data of the sensor instrumentation, to be able to judge when the information is wrong in one of the systems, and which one is the wrong one. Here could of course be a

rule written that the deterministic system, although less accurate, might be the correct one, the information could be cross check with another system (such as the visual camera), or in case there are three measuring systems, then it can be a voting scheme. In regards to the SW tools suggested, a part of the architecture could be to always have a classical algorithm (non-ML) running and evaluating the distance, parallel to the ML model (it is here assumed that the ML model would be more precise than the non-ML model, and be the reason to why it is used in the first place). The results of the two methods can then be compared for every estimation, as they likely should be in somewhat close range of each other. However, in case that they are not, the system could then use the fact that all incoming data is compared to the distribution of the previous seen dataset by the ML-model. This can give an understanding of how reliable the ML-based system is at any given moment, and rules can be made whether the overall booster control application should favour the estimation of the classical algorithm with limited granularity, or the ML-model, with its stochastic nature.

Of course, it should be mentioned that in any such system, the reaction time needed to respond to failures have to be a driving factor of the design of the safety cage architecture.

c. Lastly, an evaluation can be done for the feasibility of the system. Is it possible to have additional hardware on the mission? Would it be possible to have the additional software running on the on-board computer, performing calculations before and after the ML model? In case the visual video feed also should be used, can a ML-model be trained on this also, or if yes, is there enough data for training a good model, or is it time and money-wise too expense to create good data in the case there is not enough?

Except the HW considerations, the architecture coming out of the example above could end up looking something like shown in Figure 6-20 below. This is a fairly generic example of a safety cage architecture with parts running in parallel, but it could be imagined that the box called “safety cage algorithm” would contain the comparison function, of comparing incoming data with the past seen data, and while also comparing the output of the ML algorithm with the non-ML model, which would also be running there in parallel to the ML-Model. Based on the conclusion, a gate is implemented which decides which system to trust, which can be seen as the boxes to the right side of Figure 6-20.

It should though be mentioned, that depending on system where the AI-based SW and the safety cage SW is running, it is recommended to design base on the possible limitations of the available HW (e.g. CPU and memory budget for a spacecraft).

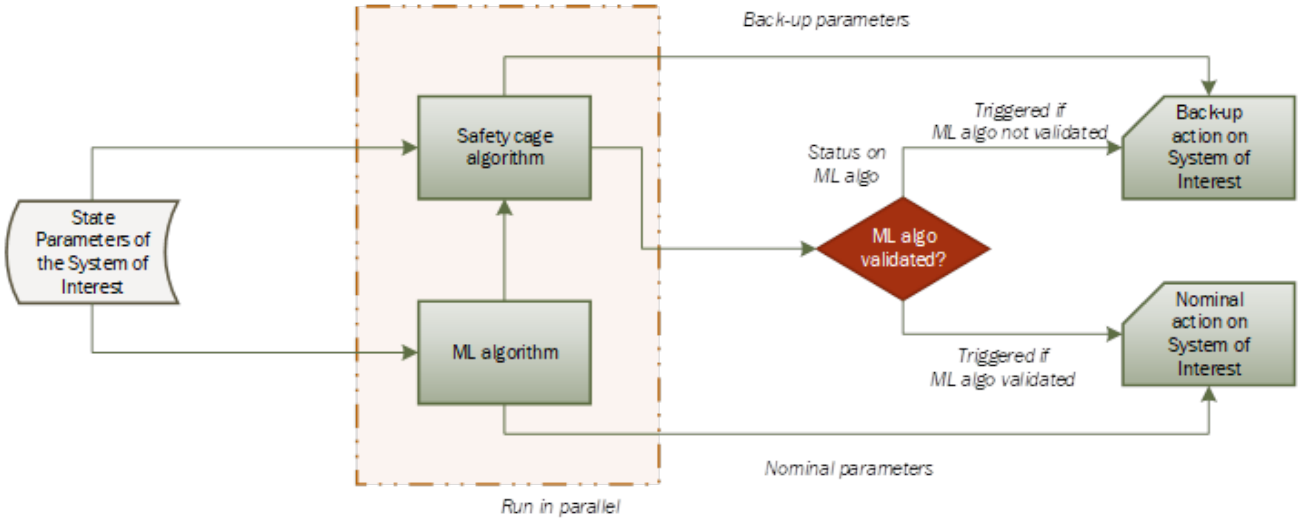

Figure 6-20: Generic example of a safety cage architecture, with processes running in parallel

Although the example above is limited in scope, as there could likely be other types of failure modes which also would be needed to be addressed as part of the architecture, which was not discussed here for simplicity's sake. However, the above example could still provide a broad idea of the process which goes into finding a safety cage architecture, and that there are potentially multiple ways of creating such an architecture.

It should also be mentioned that, although safety cage architectures are currently needed, this of course also limits the capabilities of the ML application and additional software has to be placed together with the ML applications.

However, new ML methods focusing on explainability, robustness, verifiability, etc. are under development and will allow to remove AI functions from the safety cage, enabling self-standing safety critical AI functions that can fully rely on formal verification methods.

Practical architecture recommendations

Following considerations are advised by the handbook for building an appropriate safety cage:

- Criticality of software components within the architecture needs to be defined following the guidelines described in ECSS-Q-HB-80-03 and in compliance with ECSS-Q-ST-30, ECSS-Q-ST-40 and ECSS-Q-ST-80. They are particularly relevant for the requirements about avoiding failure propagation of criticality components of different category, and the ones relative to the criticality of the software when acting as a compensating provision.
- Consider that tools can be used both before, after and parallel to the ML-model/application.
- Generally, you can avoid adding stochastic models, or models with high uncertainty to the safety cage architecture as this could introduce additional failure models. Likewise, it is recommended that the tools which are added also are qualified, which can be done the easiest if they

represent classical SW and HW.
Hence, we also recommend: “Do not mitigate ML with more ML”.

- Once an architecture is created, remember to consider if any new failure modes are introduced.
- The safety cage can run synchronously or asynchronously with the ML algorithm, depending on the time constraints. For ML-applications that perform live evaluation, such as anomaly detection and command & control applications, synchronous architecture is recommended as the default.
- Consider whether the application is an open loop or a closed loop system, and if the safety cage architecture is still adequate.
- Consider taking the possible update of the ML-model into account, and the impact it could have on the Safety cage architecture.
- For criticality category B applications, the ML algorithm and the safety cage algorithms can run on hardware platforms limiting shared resources and minimizing the risk of common failure (e.g., power supply, memory, data interfaces), and no failure propagation from the ML algorithm to the safety cage algorithms should generally be the aim.
- Whenever a safety cage is enabled and detects an invalid ML algorithm output, the values of the ML algorithm inputs and outputs, as well as the safety cage inputs and outputs can be logged in a text file for offline investigation. In case the safety cage state is recorded on a rolling window, the parameters just before the invalidation could be logged as well.
- In the case of an ML-based application which is set up to run and retrain itself continuously, it is recommended to consider what is the need from a safety cage architecture, to allow the continuous training to take place, but still control it enough to assure that the model performance is within a safe state.
- Consider whether some form of running evaluation of the model performance can be part of the safety cage architecture, as a way to avoid performance drift.
- As a best practice for the ML training, as soon as the rules driving the safety cage are defined, they can direct the learning of the ML algorithm: quite naturally, giving the direction of forbidden outputs to the model, not only will reduce the likelihood to abide the safety cage rules, but also will increase the overall convergence towards better performance of the model. For instance, the distance to the safety cage limits can be added as a penalty to the loss function (with a weight as a hyper-parameter to choose).
- Transition from the unsafe state detected by the safety cage into the safe state driven by the back-up algorithm can drive the system across intermediate unsafe states. These transitions can be carefully considered as part of the mitigation strategy.

Lastly, it is recommended to validate the behaviour of the input parameters, to assure that the safety cage is reliable compared to the intended functionality. Validation of data is discussed in section 6.4.1. Part of this validation is also to understand the device/system which produces the input to the safety cage. This

can be understood as trying to classify the system as stable vs. unstable, where an unstable system could over time create a divergence between the intended function of the ML model and the actual performance, which in turn could affect the efficiency of the safety cage architecture. It is therefore recommended when building the safe cage architecture to consider the impact of performance slippage of the ML model, and unstable input parameters to the safety cage. For unstable system periodic offset readjustment, also linked to the performance monitoring mentioned in a previous bullet above.

Safety cage building blocks and tools

In the previous sections a safety cage architecture was presented. In this section a breakdown of some of the tooling which can be used are provided, generally consisting of non-ML methods, but otherwise can be built from traditional software, algorithmic models, Symbolic AI, hardware and combinations. Hence, the safety cage takes the ML-based application and creates what is known as "hybrid AI" applications.

It should also be noted here that despite that the tools are split up, multiple tools are often used together as building blocks for the creation of the final safety cage architecture.

- **Symbolic AI/ Implementation of rules/logic gates**
  - Description: Rule-based system, mainly consisting of limited amount of hard coded rules to avoid complexity. Normally placed before or after the ML model(s).
  - Examples of usage:
    - Blocking of forbidden output value by create a binary condition which completely blocks the output of the model to propagate if triggered (based on predefined relationships with other components).
    - Implemented logic gate for when to use a reference model instead of the ML-model and reverse.
    - Setting up rules for when to use the ML based model, and when to use the redundancy scheme. Can e.g. be linked to distribution check, or valid space scenarios.
  - Application example:
    - For an AI system controlling the pressure of propulsion pipes, it should never be allowed to suggest that the pressure goes above the defined limit of the pipes. In case it does, full stop.
    - During satellite docking, the safety cage is only needed in close proximity to the target. Logic is integrated to only apply the safety cage at specific times, as indicated by a distance measuring function.
- **Voting schemes**
  - Description: Selection policy between the output of several algorithms/models

- Examples of usage:
  - Implementation of voting schemes for the choice of correct output values to be used further.
- General code implementation: often implemented right after the output of the ML-model and other non-ML software or models.
- For a voting scheme to be effective, it is recommended to consider in what circumstances the different voters can have higher or lower priority. This can be done by adding a referee function which will prioritise one of the two solutions depending on predefined rules. Application example:
  - Having a classical physics-based algorithm, or hardcoded rules, which is compared with the output of an ML-model, but depending on the confidence of the ML model for the specific data input, a voting scheme is set up between the classical algorithm and the model. An example of the issue with confidence could e.g. be a scenario with severe consequences in case of a wrong decision output of the ML model. This could be during the landing on the Moon, where the ML model is used to control the decent speed by measuring the distance to the surface. For this purpose, there might be both a ML and a non-ML system (voting scheme), with a referee that prioritizes the more robust non-ML model (having higher voting right) as the lander gets closer to the surface.

- **Monitoring schemes**
  - Description: General code implementation, setup to monitor the drift in performance. Or sudden large changes in output values, etc. for the ML-Model.
  - Examples of usage:
    - Often tied together with a rule-based system about retraining or confidence assessment of the ML-model.
  - Application example:
    - Implementation of monitoring schemes for a model in environment with limited prior knowledge of the data space to be expected.

- **Reference software**
  - Description: Classical algorithms (non-ML) or physics-based software/simulator providing the relationship between parameters. Can be placed before, after or parallel to the ML models.
  - Examples of usage:
    - Representing the physics equation-based relationship between parameters, used as part of the application, for reference of comparison to ML-model output
  - Application example:
    - A physics-based algorithm for estimating the size of the burn needed for orbit keeping is compared with the ML-model for calculating same burn, but the ML model is

taught based on feedback from burn-size and reaction from in-orbit data

- **Redundant Software system**
  - Description: Software or algorithm to be used instead of the ML-model/application in case of ML-model/application found not suitable for certain events/situations. Normally used parallel/instead of the ML-model. Can be used together with/as part of a reference software system.
  - Examples of usage:
    - In case of high-risk areas in the solution space, or areas of low confidence in the ML-model, a software can be made to take over for these areas.
  - Application example:
    - Together with out of distribution check, a ML-model is found to not be adequate for prediction of upcoming orbit events, and the application decides to use a software tool instead, which has less precision but is much more stable/reliable
- **Reference Hardware**
  - Description: Usage of hardware as reference for the ML-application. Function Placed normally as input to or part of evaluation after running the ML model.
  - Examples of usage:
    - Physical reference points for application (for example based on sensors measurements).
  - Application example:
    - The model suggests the angle of the solar panels for optimal energy generation, but thanks to some temperature sensors located on the spacecraft, it is possible to infer which side of the spacecraft is exposed to the sun, and thus coarsely verify the ML-model chose the correct orientation.
- Physical gate/filter
  - Description: Usage of hardware for blocking, limiting the ML-application. Function placed normally before or after the ML models.
  - Examples of usage:
    - Physical gate for triggering a boundary condition for the output of the ML-model.
    - A physical filter to limit output.
  - Application example:
    - Electrical Safety switch placed to physically trigger if certain event takes place (for example too high current in circuit suggested by the AI-model)

- **Distribution check**
  - Description: Check if the incoming data is within the data distribution of data which the ML-model has seen before, allowing the gauging of confidence. Function normally placed before the ML models.
  - Examples of usage:
    - Confidence evaluation, often linked with a voting scheme or a reference/backup scheme.
  - Application example:
    - For a ML-model which is expected to function initially poorly due to having limited knowledge about the data solution space. Although expected retraining over time.
- **Backup System "Man-in-the-Loop**
  - Description: Allowing a human person fully or partially, for example with the use of veto rights, to have the final decision making.
  - Examples of usage:
    - In case of high risk without code other option for safety cage tooling, or simple due to limited trust, human-in-the-loop can be introduced.
  - Application example:
    - Mission plan is calculated for a rocket launch, but a person performs the check to ensure the model is not wrong on any major things.

Real world examples of safety cages

See also [SCAGE]

The previous section introduced different tools that can be used to create safety cages. In this section a few real-life safety cage are presented.

Example from the space industry: Safety cage for distance estimation during in-orbit rendezvous.

For a mission that will include an in-orbit rendezvous with a target that is already flying, ML component for close proximity operations will be used. The ML component is based on an image recognition algorithm and will be used to determine the distance between the spacecraft and the target.

There is a risk of collision between the spacecraft and the target, which is considered a catastrophic event. To avoid this hazardous situation, it is decided to put a safety cage in place. A Kalman filter will be implemented in the application SW of the spacecraft to calculate the bounds for the estimated distance. The measurements of the distance provided by the ML algorithm will be compared to the range provided by the Kalman filter to check if the values are within expected bounds. An alarm can be raised when it is detected that the ML component starts showing unexpected behaviour, which can be used as input for FDIR to handle the malfunctioning of the ML component and either fall-back to sensor inputs only or perform collision avoidance manoeuvres. Corrective

actions on the ML component can be performed once the spacecraft is again in a nominal mode.

Examples of safety cage found outside of the space industry.

There are different applications of the safety cage principle in the literature. Some of the references use different names, such as "safety bag" or "safety monitor" but all they have common elements.

One of the first references quoting the use of a safety bag is [ELEKTRA]. This paper describes the implementation of a rule-based expert system (safety channel) to be used as a mean to check the suitability of the commands issued by a different logic (logic channel). In this case the use of the safety bag is not linked to autonomous systems, but simply seen as a mechanism to implement diversity (logic and safety channels are implemented using different programming paradigms).

Reference [SPAAS] presents the results of a study in which different alternatives are explored to protect from potential faults from AI—based system. The approach is based on the underlying assumption that these AI-based systems might be less dependable and safe than systems based on "classic" algorithms. The use of a safety bag is described, in order to monitor on-line a set of safety properties so as to authorise or not the execution of commands to the spacecraft elaborated by the autonomous software applications.

Paper [SMOF] describes a framework for the design and implementation of a safety monitor (or safety bag) as a device responsible for safety. The principle here is that there is safety channel (safety bag), being implemented following higher integrity requirements, and a control channel that is responsible to implement all the functionalities of the system. The reference describes in detail the process for the formalisation of a safety invariant, the synthesis and analysis of a safety strategy (set of rules that ensure the safety invariant), and the deployment in a real time safety monitor. An example of application is presented using a mobile manipulator robot.

Paper [AVEHIC] describes an application of the safety bag principle in the automotive domain. The purpose, as in other references, is to design a safety bag that monitors the state of the system, and moves it into a safe state in case of a hazardous situation. One of the main goals of the reference is to compare the use of different techniques: HazOp-UML and FMEA in order to derive safety requirements.

6.4.4.4 Keeping ML system validation in operations

Once the system has formally been verified and validated for operation on a space system, it can be deployed in production or used for real application. Yet, it is good practice to monitor its performance during its operational life.

Several reasons can explain why a verified and validated system could have a degraded performance during the system lifetime, compared to the performance obtained during the development and verification and validation cycle:

- Behaviour drift: the relationship between the observed inputs and the desired response from the system is no longer correct (e.g., in a degraded

case, if a propulsion engine is out of order, the attitude control system based on a full set of engines will fail to steer the vehicle properly)

- Data drift: the inputs observed by the system gradually or suddenly shift from the distribution of the test set (e.g., by wear-out of an image sensor, the recorded images become extremely dark or lose contrast below the range of illumination/contrast of the training data, therefore the computer vision algorithm fails to locate/identify target system to dock on). However even if the risk is considered and mitigated at training, monitoring remains mandatory.

To facilitate the investigation of suspicious output, it is always highly recommended to keep both the original ML algorithm resulting from the development and/or stable checkpointed versions of the algorithm, in addition to the final version used for serving where all the post-training optimization have been implemented. Therefore, it is essential to prepare the monitoring capacity of the system in operations in the very early stage of the design of a ML system, with proper logging of the inference results. The information to be recorded and the quantity of logs is tailored to the storage capacity granted to the ML system, and to the transfer capacity of the data to the monitoring team. In the case of space systems, storage capacity on board is costly and ground transfer of large amount of data is extremely expensive and often a challenge: selection of key indicators albeit sufficiently detailed to allow investigation of faulty behaviour can be integral part of the verification and validation of the ML system.

The metrics can cover:

- Inference metrics: memory load, CPU/GPU load, throughput, server load, exceptions like overflow or division by zero.
- Input metrics: histogram or repartition in the inputs space, malformed or missing values, synthetic data associated to the metrics of the training/test set.
- Output metrics: number of invalid outputs, proximity to safety cage, proximity to limits of validity.

Besides, to allow recovery actions against the issues listed above, it is strongly advised that the ML system features secure upload of ML parameters, while in operations, or even upload of the entire set algorithm + parameters.

This ML upgrade feature shouldn't come as a total and immediate replacement of the current ML algorithm, lest the new algorithm operates worse than the original one, or even locks the system. The ML system upgrade in operations can follow the same usual schemes as classical software among which:

- Shadow deployment: the new algorithm is run in parallel with the previous one; the system only uses the previous algorithm output; the results of both are compared by the development team before further deployment.
- Canary deployment: the new algorithm is applied on a small fraction of the cases; the performance is monitored, and the fraction of the inputs directed to the new algorithm is progressively increased.

- Blue green deployment: a router directs the inputs either to the previous or the new algorithm depending on which algorithm best behaves on sub-classes of inputs.

Since the exact environment of space systems is sometimes hard to simulate in the process of ML model development, it is also possible to automatically complement the training of an ML model during its operational life. Except for rather simple models, this approach isn't suitable for space systems in general, because of:

- the specific computing environment required for the training,
- the necessity to convey large quantity of data from the system to the development team to execute or even just assess the result of non-regression tests,
- the risk of self-loop, when the system favours actions that will provide data improving its local ML performance rather than the overall efficiency of the system.

It is rather recommended to transfer additional data to the development team (even down-sampled for faster transmission) so that the retraining is performed on ground with the best computing and diagnosis means.

7 Conclusion

This handbook has aimed at providing an overview of applicable resources and initial framework towards verification and validation of ML/AI systems in the space domain. Given the dynamic nature of this discipline, it is likely that updates will be required to reflect changes related to advances in AI research (such as generative AI with its unprecedented growth over the time it took to write this handbook), available of hardware that facilitates novel use-cases (as was the case with most of the deep learning achievements over the past years) and availability of data (which is of particular interest to space agencies given the dramatic increase in data volume collected from various domains).